PERCEPTUAL OPTIMIZATIONS FOR VIDEO CAPTURE, PROCESSING, AND STORAGE SYSTEMS

AMRITA MAZUMDAR

A dissertation
submitted in partial fulfillment of the
requirements for the degree of

Doctor of Philosophy

University of Washington
2020

Reading Committee:

Luis Ceze, Chair

Mark Oskin, Chair

Magdalena Balazinska

Program Authorized to Offer Degree:

Paul G. Allen School of Computer Science & Engineering

University of Washington

Abstract

PERCEPTUAL OPTIMIZATIONS FOR VIDEO CAPTURE, PROCESSING, AND STORAGE SYSTEMS

Amrita Mazumdar

Chairs of the Supervisory Committee:
Professor Luis Ceze
Associate Professor Mark Oskin
Paul G. Allen School of Computer Science & Engineering

Visual media is the dominant form of content used in modern computing systems. Advances in machine learning, virtual reality, and display form factors drive demand for richer visual experiences, putting pressure on systems to efficiently use compute and storage infrastructure. At the same time, the rapid pace of performance and energy efficiency gains computer architects depended on to meet growing application requirements has slowed. Designing computer systems to meet the requirements of modern video-based applications requires specialization in compute design, using hardware-software codesign techniques to closely optimize computer system performance for specific visual computing workloads.

This thesis uses perceptual information to optimize the design of video capture, processing and storage systems. I describe system optimizations using three classes of perceptual cues: structure (e.g., color, depth); semantics (e.g., faces, objects); and saliency (e.g., human visual saliency, neural network feature saliency). This thesis demonstrates how perceptual information can be used in hardware accelerator designs on ASICs and FPGAs, and in cloud video storage infrastructure.

To my parents.

A new sense of the notion of information has been constructed around the photographic image. The photograph is a thin slice of space as well as time. In a world ruled by photographic images, all borders ("framing") seem arbitrary… The ultimate wisdom of the photographic image is to say:

> "There is the surface. Now think–or rather feel, intuit–what is beyond it, what the reality must be like if it looks this way."

— Susan Sontag, *On Photography*

ACKNOWLEDGMENTS

Completing a PhD is a singular achievement but also one impossible to achieve alone.

I must first thank my co-advisors, Luis Ceze and Mark Oskin. To be coadvised is to be paired with two indefatigable champions with no clear model for consensus. Luis taught me to be ambitious in research impact and aggressive with my creativity. Mark taught me to be discriminating in idea selection and lighthearted in my daily work. The act of balancing their unending ideas and criticism has shaped the mold of my research forever. Thank you for advising me!

Like any PhD student at the Allen School for Computer Science, I had not only two advisors but also a cabal of faculty and industry researchers helping me along. Notably, Magda Balazinska taught me how to lift architecture research into the world of systems and how to lead research collaborations with thoughtfulness and clarity. Other mentors improved my work through significant revision, feedback, entropy, and expertise: Jon Barron, Alvin Cheung, David Gallup, Warren Hunt, Anton Kaplanyan, Visvesh Sathe, Steve Seitz.

Much as I tried to conduct a research project without anyone's help, I could not have completed a single page of this dissertation without my friends and lab mates: Armin Alaghi, Meghan Cowan, Maureen Daum, Emily Furst, Brandon Haynes, Brandon Holt, Sung Kim, Vincent Lee, Ming Liu, Liang Luo, Thierry Moreau, Brandon Myers, Jacob Nelson, Benjamin Ransford, Adrian Sampson, Pratyush Patel, Luis Vega, Mark Wyse, Eddie Yan. Very special thanks to my friends Kira Goldner, Ellie Levey, Lucy Lin, and Maarten Sap for providing company at a fortifying brunch as necessary.

Preparing a dissertation may be hard, but running the operations that allow research to be researched and innovation to be innovated is much harder. None of my work could be possible without the emotional and infrastructural support of Elise Dorough, Melody Kadenko, Amanda Robles, and Anna Wehovsky.

My arrival at the University of Washington would not have occurred without a cosmic alignment of advising from faculty at Columbia University: Steven Edwards, Martha Kim, and Shree Nayar. The CRA-W's DREU program connected me to Iris Bahar at Brown University, my first guide through computer engineering.

I am grateful to the staff of High Technology High School for introducing me to research, and Gary S. Settles for being my first faculty mentor. My original education on abstraction and interfaces came from a decade of piano lessons from Beth S. Chen.

Finally, thank you to my parents, Rekha Datta and Subrata Mazumdar, for supporting me in a great many things but especially this process.

No level of thanks could possibly be sufficient to account for Gregory T. Woolston's contribution to this work; I will not even begin to try.

CONTENTS

1 INTRODUCTION

Visual media, namely photos and videos, is a fundamental driver of computer systems applications. Compressed video data, for instance, constitutes 70% of Internet traffic and is stored in hundreds of combinations of codecs, qualities, and bitrates [2, 35]. These videos are produced by increasingly inexpensive and low-power mobile cameras, deployed at scale in everyday consumer devices like mobile phones and home sensors, as well as in city infrastructure surveillance systems, modern cars, and wildlife refuges. New domains of video production and viewing—e.g., panoramic (360°), stereoscopic, and light field video for virtual reality (VR)—further compound the production of visual media. Furthermore, improvements in display technologies and computer vision algorithms motivate not just the extensive use of cameras and video capture, but also consumption of increased volume of video content. We are not just producing *more* photos and video, but also at higher resolutions, faster frame rates, and with larger dynamic ranges of color, and distributing them to larger groups of users.

As these video and graphics applications become richer and more immersive, however, they push the limits of already-strained computer systems. For decades, computer architecture has leveraged the easy gains of Dennard scaling to provide exponential performance with little added complexity to general purpose computer processing units (CPUs) [44]. As CPU design has reached its physical limits, computer architects have looked to designing custom hardware to achieve the increased compute and memory specifications required for modern computing workloads [93, 154, 190]. By tailoring the design of compute infrastructure to meet the needs of specialized visual computing workloads, these systems can improve speed or power consumption, leveraging hardware like graphics processing units (GPUs), general purpose GPUs (GPGPUs), field programmable gate arrays (FPGAs), and custom application specific integrated circuits (ASICs).

1.1 CHALLENGES FOR VISUAL COMPUTING SYSTEMS

Visual computing applications are a natural target for specialized custom hardware: the two-dimensional image representation is easily expressed using array and buffer primitives; many image processing filters are simple convolutional kernels easy to broadcast in parallel. But the cost of data movement (both within a single hardware design and across networks to different chips, modules, or computers in a datacenter) for these large volumes of visual media for VR and video processing workloads far exceeds the benefits of tightly optimized parallel algorithms. Managing this data requires more holistic design approaches beyond parallel compute patterns and stream processing.

Fundamentally, images and video data use pixel-level information (RGB color at XY coordinates over time) to communicate higher-level information (motion, objects,

visual importance, scene meanings). However, past work in specialized image and video hardware accelerators [4, 74, 155] only optimizes hardware at the pixel level, processing every pixel uniformly and measuring tradeoffs in designs by evaluating pixel-level metrics.

While architectural challenges have motivated system designers to push performance limits by codesigning custom systems for visual media applications, this matrix representation of visual media has remained stagnant. On the other hand, many visual computing applications produce results at a frame or video segment granularity, as opposed to pixel-level results. For these applications, integrating this higher-level intent in the hardware-software codesign process introduces new opportunities for improving performance as well as new challenges in system design.

1.2 THESIS OVERVIEW

This dissertation posits that *codesigning video-focused systems with perceptual information* improves performance and energy efficiency while maintaining application quality of service. To support this, I employed techniques from the domains of hardware-software codesign and machine learning for systems optimization. Combining these computer systems and architecture methods with vision and graphics research on extracting meaningful information from 2D visual media, this dissertation demonstrates how to improve the performance of constrained video processing systems while maintaining high quality of experience.

The visual computing systems I considered operate on images and video at various stages of the visual computing pipeline: during the *capture*, *processing*, or *storage* phases, or some combination therein. In existing work, hardware-software codesign alone has been shown to improve performance in these domains (Chapter 2). My thesis builds upon this prior work to show how the use of perceptual optimizations can expand the design space and help identify new accelerator designs. The resulting systems demonstrate how to use perceptual cues to improve the power efficiency, latency, and storage capacity of video management systems, and they highlight how treating visual media at a richer granularity than matrix arrays of pixels uncovers new opportunities for optimizing system performance.

I define *perceptual optimizations* as a class of optimizations that improve system performance by targeting a quality metric or perceptual measure corresponding to higher-level meaning in the image or video, as opposed to per-pixel quality. This dissertation considers three classes of perceptual optimizations: *structure*, *semantics*, and *saliency*. To explore these perceptual cues, I used hardware-software codesign and machine learning for systems techniques to optimize computer systems at different points in the video processing pipeline.

1.2.1 *Structure*

Structural perceptual cues encode detailed information about the structure of the scene in the visual frame. For instance, simple convolutional kernels can be used to extract *edges* from the scene, an elementary form of structure to draw boundaries [50]. More complex computational photography algorithms can generate *motion*, *depth*, and *lighting* from a series of images or video [9, 12, 175]. Making use of these perceptual cues allows for applications to generate new viewpoints or lighting conditions or understand the distance between pixels in 3D space. Structural perceptual cues can be captured from complex sensors like depth or IR sensors, or synthetically generated through image processing or more complex computational photography and machine learning algorithms.

Chapter 4 makes use of structural cues to rapidly process virtual reality video from a complex camera rig. In that work, I used structural information about color and depth in captured video to design an algorithm for fast, accurate 3D-360° video generation (Chapter 4). In particular, I formulated a new data structure for processing video captured from VR camera rigs into visually pleasing 3D-360° video, and design a hardware accelerator around this data structure for real-time VR video generation. I proposed a new algorithm, the hardware-friendly bilateral solver, that enables faster runtimes than existing algorithms of similar quality. I also designed an FPGA-based hardware accelerator that utilizes reduced-precision computation and the parallelism inherent in our algorithm to achieve further speedups over our CPU and GPU implementations while consuming an order of magnitude less power. Combining the FPGA design's power efficiency with our structure-aware algorithm enabled practical, real-time VR video processing at the camera rig or in the cloud.

1.2.2 *Semantics*

Semantic perceptual cues generate information about the *meaning* of the scene encoded in the visual frame. For example, the number of objects detected by an object detection algorithm [184] is a semantic cue. A more expressive semantic cue would use an object recognition algorithm to identify each object and map the label to a bounding box or location [159]. These semantic cues provide value because they give insight into what the user perceives in the frame, beyond the colors at each pixel location. Semantic perceptual cues can be generated by crowdsourced user labeling, or, as in more recent systems for machine learning works, generated by use of computer vision algorithms on large batches of visual media [153].

In Chapter 3, I evaluated the use of semantic information in the design of hardware ASICs for an energy-harvesting camera system. That work considered not only the use of semantic cues, but also how to balance computational complexity with the accuracy and specificity of algorithms generating semantic information. I composed two accelerators—a simple machine learning algorithm and a more computationally intense one—to conduct semantic filtering for improved energy efficiency. I proposed an

organization of the hardware designs where one specialized accelerator first filters for one granularity of semantic information and then offloads processing to a more complex neural network to generate more detailed semantic data. To implement the desings, I employed several energy savings techniques such as reduced-precision functional units, low-voltage logic and memory, hardware specialization, and take full advantage of the accuracy × resource usage trade-off offered by the ML algorithms. The results showed ~100× improvement in performance and energy efficiency, enabling applications that would not be possible without a semantics-aware hardware accelerator.

1.2.3 *Saliency*

The perceptual cue of saliency encodes a spatial measure of visual importance. Within a single frame of video or an image, an algorithm could generate per-pixel saliency data to capture another dimension of information about the frame. Compared to semantic cues, which capture information about *what* a space of pixels means, saliency cues only encode the extent to which the space of pixels is relevant or important.

Traditionally, saliency in computer applications referred to human visual saliency, or what humans visually perceived to be salient [89]. Human visual saliency data can be collected using user trials with eye trackers or mouse movements, or generated automatically using computer vision algorithms.

In this dissertation, I used saliency to explore the design of cloud video processing systems and mobile video playback hardware. In Chapter 5, I used human visual saliency to optimize video compression and show the design of a storage system built codesigned with an understanding of this perceptual cue. The compression technique used a neural network to predict saliency information used during transcoding, and the storage manager integrated perceptual information into the video storage system with little overhead. The results demonstrated the benefit of embedding information about the human visual system into the architecture of cloud video storage systems.

In Chapter 6, I leveraged saliency cues in the form of foveated video content to design new mobile video decode hardware with increased energy efficiency. By observing that VR video combines small, high-resolution regions with large, low-resolution areas, I devised a new parallel, energy-efficient decode architecture. This architecture schedules work across parallel heterogeneous video decode cores in order to balance the compression of both high- and low-resolution video content. The accompanying profiling tool I designed helps VR developers explore and characterize the energy-consumption of video workloads under various perceptual compression schemes. These works demonstrated how automatically-generated saliency information from machine learning algorithms can optimize both large-scale video processing pipelines and power-constrained mobile video playback hardware.

1.3 ORGANIZATION

This chapter serves as the introductory framing for the remainder of the thesis. In the following chapter, I more deeply contextualize my work within the areas of specialized computer architectures and visual computing systems, and highlight prior work using the perceptual cues I consider.

Chapter 3 describes my work codesigning computer vision hardware for energy-harvesting cameras, balancing tradeoffs of accuracy and energy efficiency through the use of semantic perceptual information. Chapter 4 highlights another hardware-software codesign solution, applied to a 16-camera virtual reality video pipeline, using perceptual optimizations based on the structural content of distance and depth to improve the algorithm's accuracy and speed.

Chapters 5 and 6 describe saliency-optimized systems. Chapter 5 presents Vignette, a storage system and compression algorithm designed to support applying human visual saliency information to traditional video compression for improved storage and streaming bandwidth. Chapter 6 describes how new mobile device hardware can adapt for a future of saliency-optimized videos to improve performance and energy consumption.

Finally, Chapter 7 concludes with notes for future perceptually-optimized visual computing systems.

1.4 PREVIOUSLY PUBLISHED MATERIAL

This dissertation includes work published as conference papers:

- Chapter 3: Amrita Mazumdar, Thierry Moreau, Sung Kim, Meghan Cowan, Armin Alaghi, Luis Ceze, Mark Oskin, and Visvesh Sathe. "Exploring computation-communication tradeoffs in camera systems." In: *2017 IEEE International Symposium on Workload Characterization (IISWC)*. IEEE, Oct. 2017, pp. 177–186. DOI: 10.1109/IISWC.2017.8167775. URL: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=8167775&isnumber=8167743

- Chapter 4: Amrita Mazumdar, Armin Alaghi, Jonathan T. Barron, David Gallup, Luis Ceze, Mark Oskin, and Steven M. Seitz. "A Hardware-friendly Bilateral Solver for Real-time Virtual Reality Video." In: *Proceedings of High Performance Graphics*. HPG '17. Los Angeles, California: ACM, July 2017, 13:1–13:10. ISBN: 978-1-4503-5101-0. DOI: 10.1145/3105762.3105772. URL: http://doi.acm.org/10.1145/3105762.3105772

- Chapter 5: Amrita Mazumdar, Brandon Haynes, Magda Balazinska, Luis Ceze, Alvin Cheung, and Mark Oskin. "Perceptual Compression for Video Storage and Processing Systems." In: *Proceedings of the ACM Symposium on Cloud Computing*. SoCC '19. Santa Cruz, CA, USA: ACM, 2019, pp. 179–192. ISBN: 978-1-4503-6973-2. DOI: 10.1145/3357223.3362725. URL: http://doi.acm.org/10.1145/3357223.3362725

2

RELATED WORK

This document describes perceptually optimized video processing systems, ranging from energy-constrained mobile cameras to cloud video storage infrastructure. I first present a survey of related work in hardware and software support for video systems, from mobile devices to cloud video computing. I then organize relevant works based on their use of the perceptual optimizations we consider in this thesis: structural, semantic, or saliency-based. Figure 2.1 contextualizes the domain of works considered in this chapter.

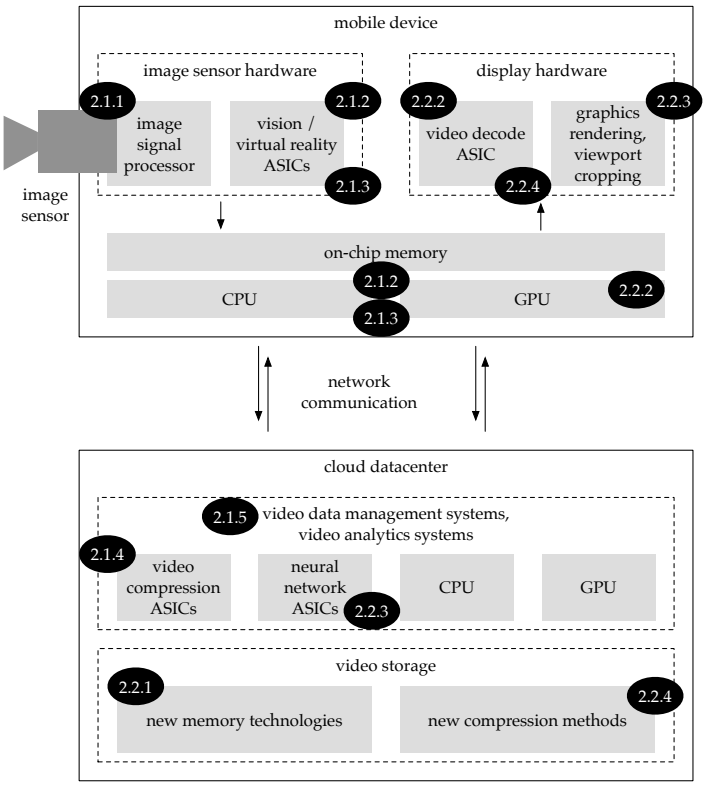


Figure 2.1: This document's related work covers research in visual computing systems spanning capture, processing, and storage. Prior work related to camera capture hardware, including near-sensor accelerators, vision-centric accelerators, and on-camera acceleration, is discussed in Sections 2.1.1,2.1.2, and 2.1.3. Related work on cloud datacenter infrastructure for visual computing is discussed in Section 2.1.4 and Section 2.1.5. A key concern for this thesis is minimizing data movement and overall data bandwidth both within the delineated platforms and across them.

2.1 SYSTEMS AND ARCHITECTURE FOR VISUAL COMPUTING

2.1.1 *Near-sensor processing for camera capture*

Image sensor data is typically captured as an analog signal and then converted to a digital signal for processing. Recent work investigated how to improve image capture efficiency by moving preliminary processing into the analog domain at the sensor node. Centeye chips [25], for instance, allow arbitrary analog computations on the sensor signals. Other work studied computing early layers of convolutional NNs at the pixel level [29, 111, 137]. In addition to the analog domain, these early compute stages can be implemented in the mixed-signal domain, using techniques like stochastic computing [5, 55].

2.1.2 *Vision-centric specialized hardware accelerators*

After visual data is captured and converted into a digital signal at the sensor node, it is sent through an image sensor processor (ISP). The rise of computer vision and computational photography has inspired a number of computer architectures to support image processing as well as more complex functionality. Flexible vision architectures [27, 36, 37, 155, 181] provide higher performance for image processing and vision applications while maintaining programmability. One commercial vision processor, Movidius's Myriad 2 processor [14], has 20 hardware accelerators supporting a number of image processing kernels as well as GPU-like accelerators and a RISC core. The Frankencamera [4] is a software-programmable camera architecture integrated with the camera design and form factor. Mobile SoCs such as TI's OMAP, Nvidia's Tegra, and Qualcomm's Snapdragon provide image processing functionality as well as programmable computation for mobile cameras [88, 143, 156].

Hardware acceleration for specific vision applications is also well-explored to provide faster performance or more energy efficiency: these include face detection on FPGAs [32, 81] and GPUs [80], and neural network vision accelerators on ASIC, FPGA, and GPU platforms [14, 19, 31, 42, 43, 48, 49, 132, 155, 158, 160, 203]. These implementations explore performance tradeoffs across parallelized configurations of processing elements, using different hardware substrates or data movement techniques to improve speed or energy efficiency.

Recent work addresses energy efficiency of mobile video compression as a critical workload as well. Finchelstein et al.evaluate the impact of parallel decoder cores on a H.264 chip design [56]. Tikekar et al.contribute a related parallel video decoder design using embedded DRAM to reduce data movement for VR devices [179]. Boroumand et al.evaluate executing video decode hardware with processing-in-memory (PIM) to reduce energy from data movement [18].

Instead of designing custom hardware to support specialized vision computation, another approach is to offload image processing computation from mobile devices to the cloud, where compute resources are more plentiful and programmability is

more flexible [166]. Forchheimer and Astrom [57] discuss in-sensor image processing as a method to reduce image sensor data to useful information by performing pre-processing. Cuervo et al. use computation offloading to accelerate complex processing on mobile devices [40]. Gharbi et al. use computation offloading techniques to rapidly apply image filters while saving energy [64]. But in recent years, the opposing case for "onloading" computation (i.e., migrating the computation back to the sensor) has grown more popular, due to the exponential growth in raw image sensor data and network communication costs as well as privacy concerns [73, 112]. Barr and Asanović observed that naive lossy compression can needlessly increase energy consumption for offloading if it is more expensive than computing on-device [11]. Han et al. use sensor-based filters to reduce the size of data to be processed on-device to make onloaded computation feasible [72]. Our hardware-software codesign approaches explore the tradeoff space between offloading and onloading in the context of highly constrained camera systems.

2.1.3 *Hardware accelerators for virtual reality video capture*

There are many ways to capture and render VR video, but each method presents unique challenges. Light fields [108], a type of image that conveys information about the flow of light in a scene, are the most general and immersive solution to VR imagery. Proposed light field-based capture systems, however, require an enormous amount of input and output data, and rendering on the client side is compute-intensive. While impressive results for light field images have been demonstrated in VR [84], video is a greater challenge.

Other solutions for immersive video viewing, such as free-viewpoint video [23] and concentric mosaics [169], are also challenging to process and display using standard video formats, resulting in systems that are not yet stable enough to motivate hardware support. In contrast, the Jump VR video system [9] is designed to be practical to compute, edit, and stream. Processing the 16 camera video array requires computing optical flow between images from each adjacent pair of cameras and then interpolating the images to produce the *omnidirectional stereo* projection [151]. The output is a pair of equirectangular spherical video streams, one stream for the left eye and one stream for the right. Anderson et al. [9] employ a bilateral-space solver for optical flow to efficiently produce high quality edge-aware flow results that are well-suited to omnidirectional stereo image interpolation. They observe that the majority of rendering time is spent running the bilateral solver. Modern systems using similar pipelines require users to upload camera streams to a cloud service or high-performance computing system—this workflow impedes real-time applications such as live VR video streaming. While real-time hardware systems for processing VR video are becoming commercially available [66, 177, 183], these existing solutions provide either live panorama processing or stereoscopic 3D, not both.

Most similar to the design discussed in Section 4 is that of Rithe et al. [160], who designed a low-power reconfigurable processor for bilateral filtering. Their design differs by implementing the splatting and slicing operations in hardware, but it can

only perform streaming bilateral filters and does not support repeated iterations of filtering in-memory, a necessity for fast bilateral-space optimization.

2.1.4 *Video streaming and storage systems*

The rise of video sharing applications has driven significant recent work in processing and storage systems for video content. An early example, the Stony Brook Video Server, codesigned its video streaming infrastructure with disk storage to deliver real-time video streaming for early Internet architectures [182]. Since then, video systems have specialized for subdomains like social media, entertainment, and video streaming. Social media services like Facebook or YouTube distribute user-uploaded content from many types of video capture devices to many types of viewing devices, typically serving a small number of popular or livestreamed videos at high quality and low latency, as well as a long tail of less popular videos [51, 176]. These workloads motivated the introduction of custom media storage infrastructure and fault-tolerant frameworks for processing video uploads at scale [16, 85, 116, 134]. Entertainment platforms like Netflix and Amazon Video have smaller amounts of video data than social media services, but incur significantly more network traffic to distribute videos broadly. These services maintain user experience by transcoding videos at high quality for a range of heterogeneous devices and bitrate requirements, tailoring encode settings by title, streaming device, and video scene [1, 98, 121, 139].

Recent systems work has proposed new designs for datacenter-scale video transcoding and streaming infrastructure, specifically optimized for the large data bandwidth and many streaming endpoints typical of modern video workloads [58, 59, 114, 120]. These systems codesign compute and network layers with software implementations of typical video transcoding steps to better leverage datacenter resources, working within the bounds of existing video compression or streaming application requirements.

2.1.5 *Cloud video management and analytics*

While systems for optimizing cloud-scale transcoding are well studied, more recent focus on pushing video content through complex analytics and processing pipelines has motivated a resurgence of interest in video database management systems (VDBMSs). Early systems focused primarily on supporting image search queries and content-based image retrieval, with workloads on the order of a thousand images [24, 28, 46, 92, 145]. In comparison with the earlier VDBMS designs, these more recent VDBMSs have dramatically increased their supported workload sizes and domains supported, such as for a wide range of deep learning for video analytics and computational photography workloads, as well as compressed video and light field data formats. These VDBMSs include Scanner [153], LightDB [75], NoScope [95], BlazeIT [94], DeepLens [100], Chameleon [91], Focus [82], Optasia [117], Tahoma [8] and VideoStorm [199]. Prior work has also evaluated using hardware acceleration [114, 116, 120] in the cloud to improve video processing performance.

2.2 PERCEPTUAL OPTIMIZATIONS FOR VISUAL COMPUTING

2.2.1 *Approximate visual computing*

Approximate computing research seeks to trade off computation and energy consumption for quality. The key value proposition is that many application domains, such as visual computing, can tolerate small drops in quality for benefits in speed or energy efficiency [130, 164]. While much of the literature considers image and video processing a natural domain for evaluating general-purpose approximate computing techniques, the metrics for evaluation are restricted to per-pixel quality metrics, as opposed to higher-level quality information [69, 104, 132, 164]. A notable exception to this is approximate computing work targeted at machine learning applications, which measures precision and recall for frames or different video segments [132]. Approximate computing techniques have been specifically applied to video processing [192] and storage [90] in support of conventional video compression algorithms, as well. While these techniques are in the spirit of trading off quality for visual computing performance, they either focus only on pixel-level optimizations or are more general-purpose and coarse than the perceptual optimizations considered for this thesis.

2.2.2 *Structural video data optimization*

Structural perceptual information refers to the physical structure of the visual data (e.g., spatial location, depth perception, color spaces). While these attributes exist or can be derived from traditional 2D visual data structures, recent work in computational photography and vision has built algorithms operating on new data representations that hold these structural elements in equal importance to pixel-based information.

The bilateral grid is an illustrative example of structural data. The bilateral grid encodes color, position, and depth for pixels in an N-dimensional grid data structure. Through this encoding, bilateral filters can be executed fastly and cheaply, to smooth images while maintaining sharp edges, which would have been obscured in a traditional two-dimensional blur. This structural information allows for algorithms to process visual data while maintaining this perceptual fidelity. The original bilateral grid was proposed as a solution for fast, parallelizable bilateral filtering on GPUs [30]. Towards more hardware-efficient execution on GPUs, Yang [194] proposed a hierarchical bilateral filter technique, but their approach is too error-prone for the high-resolution content used in high-resolution or virtual reality video today.

Another example of structural perception optimization leverages perceptual color and light tradeoffs in viewing digital content. Recent work like Crayon [172] and Ishihara [173] proposes replacing power-hungry colors with perceptually similar ones to achieve better power consumption on mobile displays, while ShutPix [193] proposes to turn some pixels off entirely. These structure-based optimizations use existing information (e.g., edges, depth, color space) in a video frame to uncover opportunities for perceptual optimization, rather than using newly-generated perceptual information.

2.2.3 *Semantics-based video optimization*

Semantics-based video optimizations concentrate effort on areas where objects or regions of defined importance are identified in video frames. Many of the VDBMSs proposed to support deep learning or video analytics employ some form of semantic-based optimization to efficiently distribute datacenter resources [82, 91, 94, 95, 199]. Specifically, they combine the use of multiple classification algorithms to more intelligently distribute computing effort to video frames that are determined to have interesting content. BlazeIt [94], for instance, calls their coupling of simple, efficient classifiers with complex, precise ones "scrubbing queries", and Focus [82] uses the technique to build up an "approximate index" of relevant videos. Recent work in virtual reality video streaming [105, 107] demonstrates another example of semantic optimization, using the object recognition classes output by neural networks to drive video transcoding or rendering effort.

2.2.4 *Saliency-based video optimization*

Saliency-based video optimizations include information about visual importance. These saliency predictions can be generated from machine learning algorithms or hand-annotated, and targeted at specific domains of saliency: human visual saliency [103], 2D graphics and visualization saliency [17, 21], or machine learning feature saliency [146, 170, 196, 204].

Early work in human visual saliency prediction improved its accuracy [103, 110], the speed of prediction [68, 70, 205], or compression efficiency [68, 71, 118, 171, 205]. These existing solutions for video compression require custom versions of outdated compression codecs or solving costly optimization problems during each transcoding run. Sitzmann et al. [171] observe the impact of leveraging saliency for VR video compression and identified key perceptual requirements for future saliency-video codesign work.

Foveated compression tailors video compression specifically to the attention driven by a single eye's gaze and it demonstrates significant bitrate savings [6, 63]. Google's foveation pipeline uses multi-resolution compression [15]. Furion leverages parallel cores to achieve fast tile-based perceptual rendering for games, but their decode is implemented in software and limited to four available mobile CPU cores [102]. Tile-based streaming has become a popular method to deliver video content adapted to a VR viewer's head location [113]. Facebook recently described an AI-based decoder for sparse foveated video, requiring machine learning infrastructure to decode frames [97]. More recently, streaming bandwidth requirements for 360° and VR video were shown to be optimized by decreasing quality outside the VR field-of-view [47, 76, 115, 161]. These foveated and region-of-interest (ROI) coding techniques could be considered either saliency-based optimizations.

3

NEAR-SENSOR FACE DETECTION

This chapter considers the design of a processing pipeline for an energy-constrained camera, the WISPCam [135], a battery-free platform which operates entirely on harvested RF energy. We characterize a continuous vision pipeline using face authentication (FA), a core workload in user-centric continuous mobile vision systems. In these systems, a camera captures image frames at a continuous frame rate, and an on-node processor performs face recognition on each frame to identify a user. These applications are common in biometrics and vision-based personal analytics. In the design of this system, we investigate ways to use perceptual information

The WISPCam-based system captures an image at 1 frame per second (FPS) and transmits it over RF, powered by an internal capacitor with harvested RF energy. We examine how leveraging progressive filtering hardware can dramatically reduce the power consumption of such a system and enable continuous face authentication at low cost. We construct our FA pipeline around a neural network (NN) accelerator performing face authentication, as shown in Figure 3.1.

The pipeline has one primary block, the NN, and several optional blocks. Given the WISP's energy constraints, we design the accelerators to be integrated on-chip with the camera sensor to minimize energy, and processed the pixels streaming through the CSI2 camera serial interface. Co-designing our accelerators with the streaming camera interface helps optimize our design.

We analyze the algorithmic and microarchitectural choices that can improve the efficiency of such a pipeline while minimizing the impact on accuracy. We first discuss the hardware accelerators required for the camera pipeline, presenting their microarchitectures and the tradeoffs we investigated in each design's algorithm and hardware implementation. We then evaluate them together on a real-world face authentication workload using real video we collected. Our evaluation discusses the performance and power improvements, specifically exploring the tradeoffs in power consumption arising from considering the total cost of computation and communication.

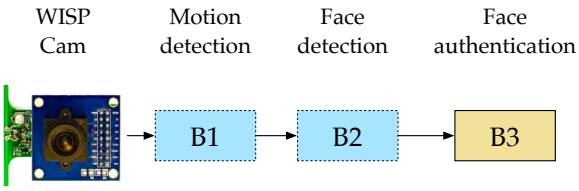


Figure 3.1: Face authentication with battery-free cameras.

3.1 BACKGROUND: FACE DETECTION & MACHINE LEARNING METRICS

3.1.1 *Viola-Jones algorithm for face detection*

The Viola-Jones (VJ) face detection algorithm is a state-of-the-art computer vision algorithm for fast, accurate face detection [184]. It detects faces by scanning a window of fixed size across the image, looking for features of a face, such as eyes and nose. If enough of these features are found, then the window at a particular location in the image is identified as a face. To account for faces of different sizes in an image, the scanning window is scaled and passed over the scene multiple times. The face detection process at each window is independent of windows at other positions and scale sizes.

The Viola-Jones algorithm is well-known because of its simplicity and efficiency, and continues to perform well against more complex algorithms [122]. The algorithm's efficiency stems from three factors: (i) simple feature designs, (ii) use of an efficient encoding and data layout (i.e., integral images) that reduces data accesses, and (iii) the cascade classifier structure.

RECTANGULAR FEATURES. The features used in the VJ algorithm are a weighted sum of rectangular pixel areas. These features are referred to as *Haar-like features* because of their similarity to Haar wavelet functions, which are natural basis functions for encoding differences in average intensities between different regions [148]. VJ uses three rectangular features (more recent work has extended the set of features, but we focus on the original three).

A two-rectangle feature is defined to be the difference between the sum of the pixels within two rectangular regions where the two regions have the same size and shape and are horizontally or vertically adjacent. Three- and four-rectangle features are constructed in a similar manner. In a given 20×20 window of image pixels, the exhaustive set of rectangular features is 160,000. For fast classification, the VJ framework employs the AdaBoost [61] learning algorithm to choose a small set of critical features for detecting faces.

CASCADE CLASSIFIER STRUCTURE. While feature computation is fast and efficient, an accurate classifier executes a large number of these computations per window. The VJ algorithm reduces the computation overhead by ruling out "easy-to-dismiss" image windows quickly. This is done by constructing small classifiers from weighted combinations of features, and then grouping these classifiers into successively more complex stages in a cascade structure (Figure 3.2).

The first classifier stage searches the image window for a small number of critical features that exist in all the face images. If the window contains all the features, it is processed by the next stage in the sequence of classifiers. Each stage in the sequence is more complex than the last, and if any stage rejects the image window, the classifier stops processing on the window. This cascade classifier structure is similar to a degenerate decision tree.

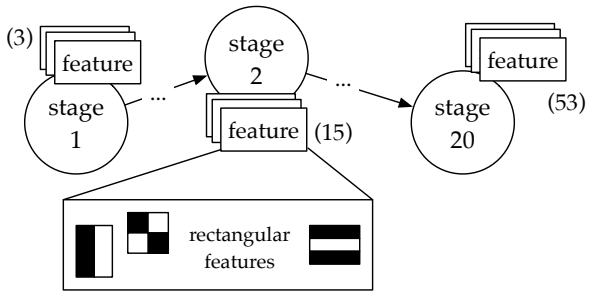


Figure 3.2: Features grouped into cascading stages.

INTEGRAL IMAGE. Computing the rectangular features involves summing across the pixels in a rectangular area, and taking the average of those pixels. A data structure called an *integral image* expedites these sum calculations. In this data structure, the value at any pixel position is the sum of all the pixels value above and to the left of it.

This representation allows us to obtain the sum of any rectangular subset of the image in constant time:

$$I_\Sigma(x, y) = \sum_{\substack{x' \le x \\ y' \le y}} I(x, y) \tag{3.1}$$

The sum of the pixels in within the rectangle defined by x_0, x_1, y_0, y_1 can be obtained by

$$\sum_{\substack{x_0 < x \le x_1 \\ y_0 < y \le y_1}} I(x, y) = I_\Sigma(x_0, y_0) - I_\Sigma(x_1, y_0) - I_\Sigma(x_0, y_1) + I_\Sigma(x_1, y_1) \tag{3.2}$$

Furthermore, constructing the data structure is also efficient, because it can be calculated incrementally. If all the value of $I_\Sigma(x, y)$ are know for the points left and above (x, y), we can calculate $I_\Sigma(x, y)$ by

$$I_\Sigma(x, y) = I(x, y) + I_\Sigma(x, y - 1) + I_\Sigma(x - 1, y) - I_\Sigma(x - 1, y - 1) \tag{3.3}$$

3.1.2 *Accuracy metrics for machine learning applications*

Precision and recall are widely used accuracy metrics used in computer vision applications to understand the quality and quantity of classifier results. For face detection, the goal is to correctly identify labeled face windows in an image as faces—these outcomes are *true positives*. A *false positive* is a classifier identifying a non-face window as a face, and *false negatives* occur when a classifier misidentifies a face as a background window. *Recall* is the ratio of true positives to all actual positive elements (i.e., how many of the actual faces were detected as faces), and *precision* is the ratio of true positives to all identified positive elements (i.e., how many of the faces detected were actually faces). A perfect classifier achieves both precision and recall of 1.0.

We can unify precision and recall into one metric, an *F_1 score*, which quantifies accuracy as the harmonic mean of precision and recall. This combined metric gives

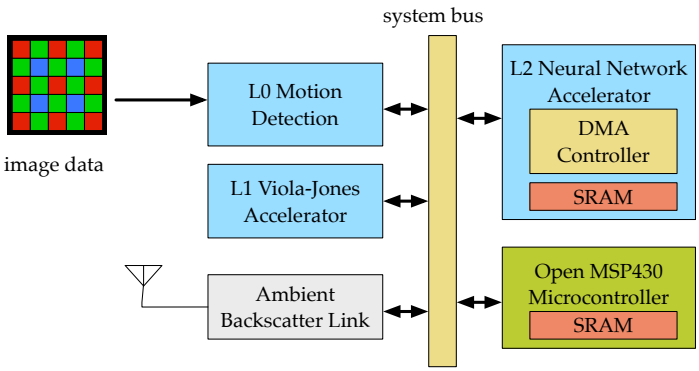


Figure 3.3: OpenMSP-based system overview.

a good first-order assessment of a design choice's impact on "accuracy", but because the F_1 score weights precision and recall equally, the score may skew towards the metric that performs worse, obfuscating the impact of an optimization on the individual metrics.

3.2 HARDWARE ACCELERATION FOR ENERGY-HARVESTED CAMERAS

A generic mobile sensor node usually consists of one or several sensors followed by analog-to-digital converters (ADCs), processing elements and wireless communication interfaces. We focus discussion on energy constrained systems, such as the WISP [163], a battery-less system that harvests energy from RFID reader signals to perform sampling, processing and communication tasks. Our ideas, however, also apply to battery-powered systems with small footprint and very long battery-life (e.g., home automation/alarm systems, and other camera-based sensor nodes).

Our case study application supports a camera platform for continuous face detection and authentication. The baseline system for our implementation is the WISPCam [135], a good example of a computationally-demanding workload that would not be achievable on a general-purpose microcontroller. An overview of the resulting system is shown in Figure 3.3.

The WISP5.0 is based on the MSP430FR5969 microcontroller with a low-power 3D ADXL362 accelerometer. The WISPCam extends it with a OV7670 camera module that can capture a 176×144 pixel gray scale image and store it in memory with a 10mJ energy budget.

We envision the described face detection accelerator to be coupled with a neural network accelerator and microcontroller and embedded in a single chip, where the image sensors and processors are fabricated on the same die. This allows low-cost access to the sensor data, since the cost of data movement is significantly reduced. One can use low-power circuits to perform motion detection. This means that a frame is captured only when motion happens, yielding a significant energy reduction. The motion detection circuit in [33] can be seen as a Level-0 (L0) accelerator that filters the frames, before sending them to the L1 accelerator for face detection.

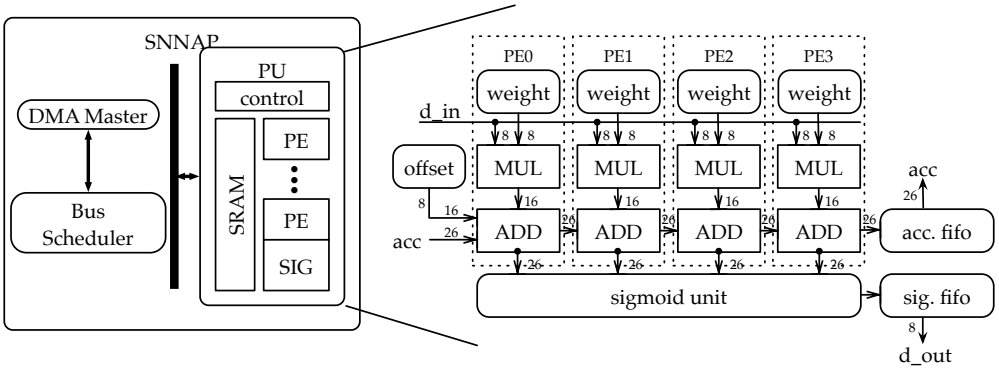


Figure 3.4: NN microarchitecture and processing element details.

```
for x in range(0,image_width):
  for y in range(0,image_height):
    faces += classifier_kernel(x, y, window)
    window = window * scale_factor
    if window > max(image_width, image_height):
      exit()
```

Figure 3.5: Nested loops of classifier invocations for VJ.

3.2.1 *Neural network face authentication*

NNs are considered state-of-the-art for image inference [7], so for our face authentication task, we employ a systolic NN design, based on SNNAP [131, 132]. We train the NNs with Fast Artificial Neural Network Library (FANN) [138], a popular open-sourced NN library. Our NN microarchitecture uses a single processing unit with multiple processing elements. We found that,because our face authentication pipeline has wide layers, this design configuration presented enough data parallelism to keep functional unit utilization high for a single processing unit. Figure 3.4 shows the datapath of a processing unit composed of four 8-bit processing elements (PEs).

3.3 IN-CAMERA DETECTION USING VIOLA-JONES

3.3.1 *VJ algorithm tradeoffs.*

We first tune the parameters of the VJ algorithm in search of the optimal point in the energy-accuracy tradeoff space. Recall that the Viola-Jones algorithm scans a fixed-size window across an image, evaluating a classifier at each x-y location, and then scales the window and repeats the search for larger-sized faces. A high-level description of this algorithm is listed in Figure 3.5.

VJ classifiers are structured to minimize computation in non-face windows: executing a full positive face detection incurs a larger computation cost, whereas deciding early that there is no face has a much lower cost. The parameters of *window scale factor*, the

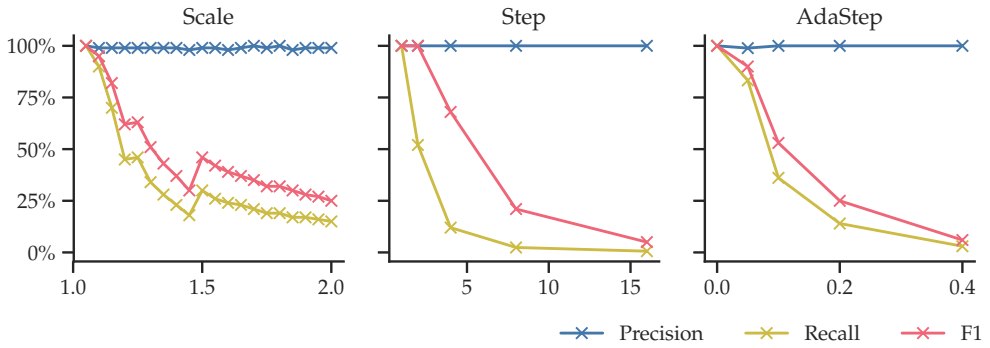


Figure 3.6: Impact of VJ parameters on relative accuracy.

size of the window scanned looking for a face, and *window step size*, the space between windows evaluated over the image, affect the number of kernel invocations, as shown in Figure 3.5.

WINDOW SCALE FACTOR. To find faces of multiple sizes, the window (and features evaluated) are scaled by a scaling factor. This scaling factor affects the number of iterations over an image the classifier must perform, and thus can greatly impact energy cost. Viola and Jones use a scale factor of 1.25 in their original evaluation, while the OpenCV implementation uses a scale factor of 1.1. For our implementation, we considered scale factors between 1.05 and 2 and explored how they impacted accuracy on a fixed-size image.

As shown in Figure 3.6, increasing the window scaling factor impacts recall but not precision, because the number of missed faces grows while maintaining a low false positive rate. In this experiment, accuracy is normalized to the smallest window scale factor, 1.05.

PIXEL STEP NUMBER. The rectangular features in a cascade classifier are averages over pixel intensities, and are thus insensitive to small changes in translation and scale. Consequently, an implementation of the algorithm can safely skip positions in both the x and y direction while maintaining accuracy. This is especially true in larger window scales, where shifting the window by a few pixels does not largely impact the results for a single feature. To demonstrate, we explore the impact of pixel step size on accuracy in Figure 3.6, normalizing accuracy to a 1-pixel step size. We find that increasing the window step number has little impact on precision, but dramatically reduces recall.

We also evaluate stepping the window position adaptively, as a small percentage of the window size. For the smallest 20×20-pixel window, we use a pixel step number of 1, 5% of the window size. The step size of the window grows linearly as a percentage of the window size. Figure 3.6 shows that adaptive stepping the window as a percentage of the window size retains higher recall than a constant step size. Accuracy here is normalized to the step percentage of 2.5%. This parameter selection results in 86% less invocations of the VJ classifier and no loss in accuracy for our real-world workloads.

3.3.2 *Classifier structure*

The structure of the classifier, or the number of stages and features in each stage, impacts the energy efficiency of the architecture as well as the accuracy of the classifier. Prior work in hardware implementation of Viola-Jones has used classifiers ranging from 5 stages with 5 features per stage to 30 stages and hundreds of features per stage. We trained cascade classifiers at a range of configuration points with OpenCV and face images from the AT&T Database of Faces [162], and evaluated their impact on accuracy, storage, and frame rate. The results are compiled in Table 3.1.

Stages	Features/Stage	Precision	Recall	Storage (KB)
5	5	0.155	0.146	1.797
5	50	0.047	0.796	6.578
15	10	0.043	0.641	10.102
10	33	0.099	0.913	17.445
15	50	0.326	0.854	33.867
20	50	0.52	0.757	50.781
22	500	0.638	0.806	58.812
30	50	0.731	0.66	80.32

Table 3.1: Accuracy and storage for Viola-Jones classifiers.

ACCURACY. Smaller classifiers are cheaper but lead to reduced accuracy for the overall system. Deploying a small classifier configuration such as Classifier 5×5 seems reasonable because of its small storage size, but its low precision and recall deliver less gains than using no accelerator at all. Increasing the number of stages and features improves the classifier's overall accuracy, resulting in better system efficiency. In Table 3.1, Classifier 10×33 has the best recall, but its precision is low, and its execution would not filter out many false positives. Classifier 20×50, however, reduces the false positive rate by 50% with only a 15% loss in precision.

ENERGY-ACCURACY TRADE-OFF. The average energy expended to evaluate an image window is a function of both the hardware design and the accuracy of the classifier being evaluated. Small classifiers that have very few features per stage, like Classifier 5×5 from Table 3.1, have very low recall rates. Low recall rates means they detect less faces, and, consequently, these classifiers display low energy consumption. Improving recall by adding more features to a classifier results in more energy-hungry classifier execution. As recall improves, however, it becomes difficult to maintain high precision. Classifiers with lower precision, which classify more false positives, will spend more energy overall than those with higher precision. This effect is demonstrated in Figure 3.7.

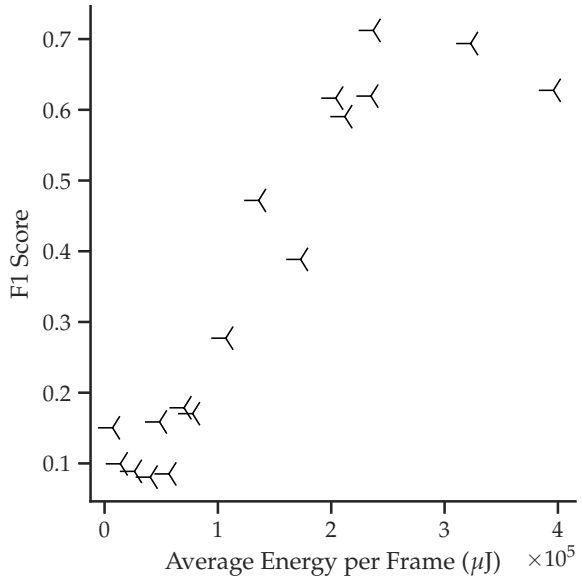


Figure 3.7: VJ energy-accuracy trade-off.

3.3.3 *VJ streaming microarchitecture.*

Many VJ accelerators exist in the literature for different application domains and power envelopes. The key observation that facilitates sub-mW power consumption for our accelerator is to process the data in a streaming fashion, leveraging the nature of the pixel stream. Designing an accelerator in the camera node allows us to process the data in the flow of the input pixel stream, rather than repeatedly fetching blocks from a memory. By co-designing the accelerator to work with the established dataflow, we reduce storage costs, reducing power consumption. This also allows us to reduce latency in delivering results to the next step in the pipeline—rather than blocking for a whole frame, the accelerator can deliver partial frame windows with faces as they are processed.

A high-level block diagram of the microarchitecture is shown in Figure 3.8. The accelerator consists of two functional modules: (i) the integral image accumulator, which transforms the image for easy feature extraction, and (ii) the cascade classifier implementation, which evaluates features and computes the classifier result for an image window.

INTEGRAL IMAGE PROCESSING. The integral image is computed by summing the pixels above and to the left of a given pixel. In our design, we compute the integral image iteratively using two row buffers—one to store the previous row and one to store the current row. By maintaining a cumulative row sum in the current row buffer and the previous integral image row it is possible to compute the integral image with minimal storage.

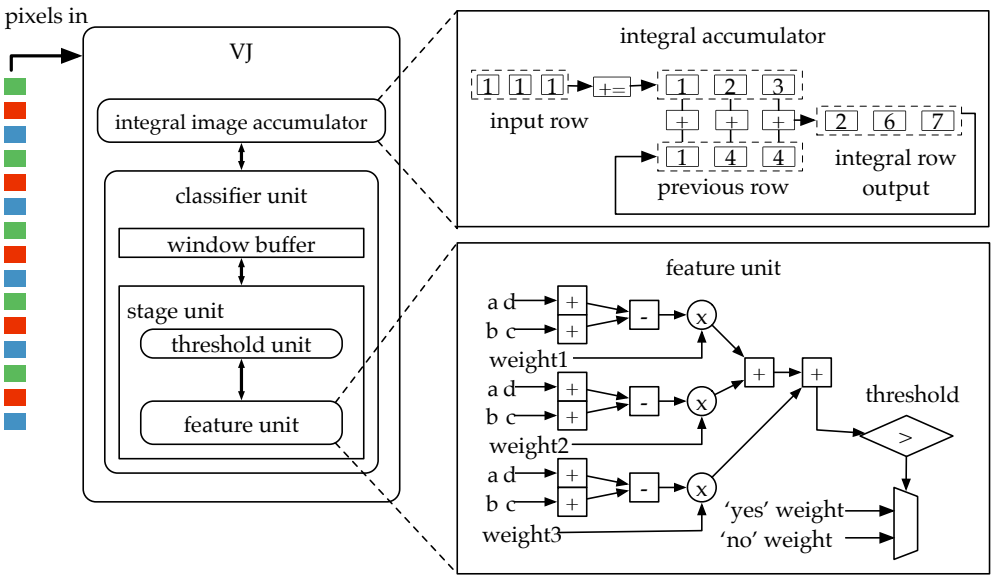


Figure 3.8: Proposed Viola-Jones accelerator architecture.

STORAGE REQUIREMENTS. Larger classifiers require more storage for the many feature and stage weights used. We use small ROMs to store feature and stage weights, and use 65 KB of storage in our design, split across two units, a 1KB ROM for stage data and a 64 KB for feature computation. This capacity can accommodate all but one of the classifiers in Table 3.1.

BIT-WIDTH REDUCTION We convert all floating-point computations to fixed-point to reduce overall energy dissipation. The data-path for computation on integral image pixels is already large, because 18-bit pixels are required to store an integral image from 8-bit sensor image. To mitigate the impact of this large data-path, we reduce the bit-width of feature and stage weights to 16-bit with negligible accuracy loss.

By computing the integral image for a whole image while buffering just two rows, we use significantly less storage than if we had buffered the whole image to compute the integral image result: for our WISPCam workload, we use less than 1kB to hold the necessary rows for integral image output, whereas buffering the whole image would require a 57kB buffer. We also included a unit to transform face windows from integral image form to standard images before they are transferred to later stages of a camera pipeline.

3.4 EVALUATION

Table 3.2 describes the details of our evaluation. We compared the performance and energy of our accelerator configuration against a software baseline running on an MSP430, a low-power microprocessor on the WISPCam. We wrote an NN micro-benchmark trained for face authentication, and emulated execution on a synthesized OpenMSP430 design. We then compared the performance of the NN and face detection (FD) acceler-

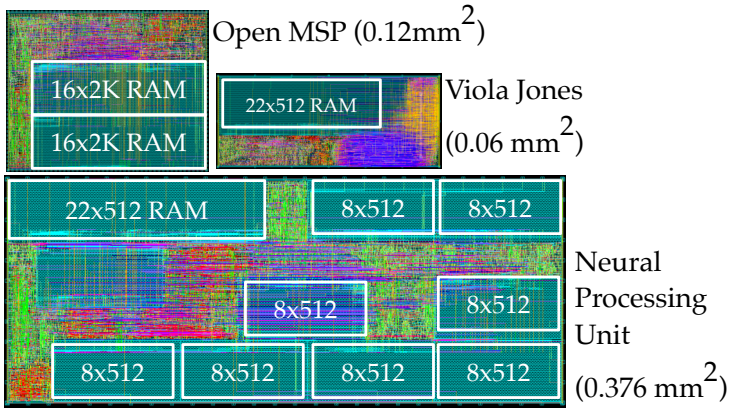


Figure 3.9: Layout shot of all three designs

ators. To obtain accurate energy estimates, we measure the initiation interval of the hardware pipelines in functional simulation and use power dissipation numbers derived from PTPX to produce energy estimates.

Table 3.2: Full face authentication system parameters.

Experimental Parameters		**VJ Accelerator**		**NN Accelerator**		**OpenMSP430 [147]**	
Technology	65nm TSMC GP HVT	Memory	1kB frame buffer	Memory	512kB IM, 4kB DM	Memory	2kB IM, 2kB DM
Tool Chain	Synposys	Datapath	16-bit custom logic	Datapath	8×8-bit PEs	ALU	16-bit with multiply
C Compiler	TI msp430-gcc [87]	Cascade	10×33	Sigmoid	256B LUT	Power	181µW
SRAM	ARM compiler + derivation	Input	176×144 pixel stream	Input	400-pixel image window	Area	$0.12mm^2$
Vdd	Low-voltage, 0.7V	Power	337µW	Power	393µW		
Freq	27.9MHz	Area	$0.06mm^2$	Area	$0.38mm^2$		

3.4.1 *Physical implementation and optimization*

We implemented our design in Verilog, synthesized with Synopsys Design Compiler using the TSMC 65nm Gplus Standard VT library, and performed place-and-route with Synopsys IC Compiler. SRAM macros are generated with an ARM Advantage memory compiler. A layout shot of our system components is shown in Figure 3.9. To find the optimal system voltage, we derive the static and dynamic components of the total system energy and select the minimum-energy voltage that still satisfies system performance requirements.

In our evaluation, system requirements constrain the baseline choice: the MSP430 is the only processor operating within the power constraints of current energy-harvesting systems. Higher-performance embedded processors, while able to execute our application with better performance than an MSP430, were limited by the power budget required. In prior NN-accelerator work, the most appropriate comparison is ShiDianNao [42], which also operates at ~300mW, still significantly higher-power than our design.

COMPARING VIOLA-JONES AGAINST NEURAL NETWORKS FOR FACE DETECTION
We compared the performance of both Viola-Jones and neural network accelerators on the same task: face detection. While neural networks offer flexibility, which make them compelling to solve a myriad of classification problems, but the specialization of the Viola-Jones accelerator results in better performance and accuracy.

For the face detection application, the Viola-Jones accelerator presents a 2.1× energy improvement compared to running a neural network. On top of that energy reduction, Viola-Jones offers much improved classification performance. Specifically we found in practice that Viola-Jones had substantially better false-positive rate than the neural network which has a more approximate nature. The efficiency benefit of the Viola-Jones classifier can attributed to its ability to spend little energy on the common case, when a window contains no faces.

Figure 3.10 shows the distribution of features across windows in a typical 176×144 image with a single face – only 0.01% of windows need to compute the full classifier.

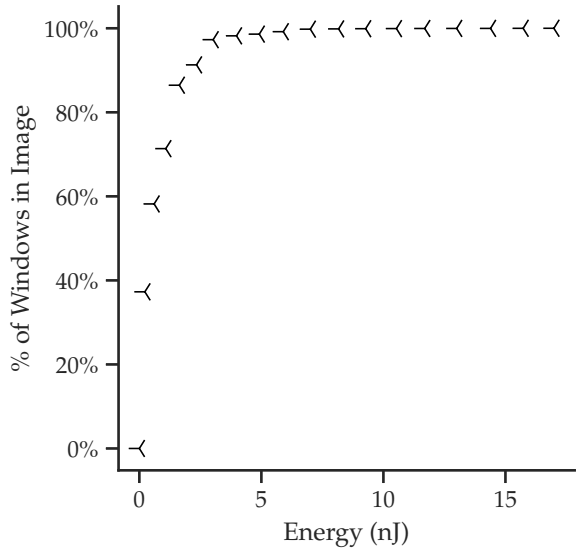


Figure 3.10: VJ energy CDF over all windows in an image.

3.4.2 *Tradeoffs in computation and communication.*

To improve energy efficiency, we evaluate hybrid approaches, which first use VJ-based face-detection to filter out the windows that do not contain faces, to only feed the small fraction of frames that do contain faces to the neural network that performs face identification. This approach takes advantage of the efficiency of the Viola-Jones accelerator, to reduce the cost of performing face identification with a neural network accelerator.

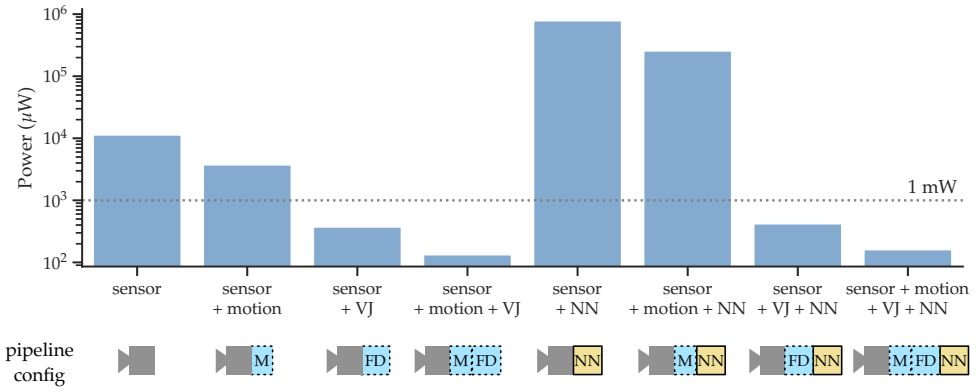


Figure 3.11: The total power for different ASIC accelerator configurations is typically dominated by either computation or communication power. The lowest-power solution uses the motion and face detection filters and offloads the NN.

METHODOLOGY FOR EVALUATING REAL-WORLD WORKLOADS. To consider the computation and communication power of our accelerators and the CPU baseline, we evaluated all configurations of the face authentication pipeline on experimentally-collected videos of real-world workloads. The dataset contains video scenarios we crafted to be representative of mobile face authentication in the wild, primarily surveillance-style security videos of a lab and a videos from a wearable-style device with an always-on camera. We collected seven videos in total, each 60-75 seconds in length: three for the security monitoring workload and four for the wearable-in-action workloads. To match the system design requirements of an energy-harvesting platform, we captured and processed the video at 1 FPS.

Using these benchmarks and the power/performance characteristics of our accelerators, we derived the input bandwidth and resulting power consumption for each computational block in every configuration of the system. To derive the cost to offload image data, we used previously reported communication power numbers [135].

COMPUTATION-COMMUNICATION TRADEOFFS IN REAL-WORLD WORKLOADS. Figure 3.11 shows our results for the full system of image sensor capture, motion detection, FD, and NN face authentication, comparing total power consumption on combinations of the ASIC designs described in this section. Because the CPU could not compute the face recognition kernel on even one 400-pixel window at 1 FPS, we did not consider the results in this section. The configurations including CPU face authentication consume 2-5 orders of magnitude more power overall than the full-ASIC design we implemented.

Figure 3.12 shows the detailed power breakdown between computation and communication for the full system in silicon. The computation power is the sum of power at that block and the processing blocks preceding it, and the communication power is the cost to transfer the output of that block. As expected, the computation power increases as more blocks process the data while the communication power decreases,

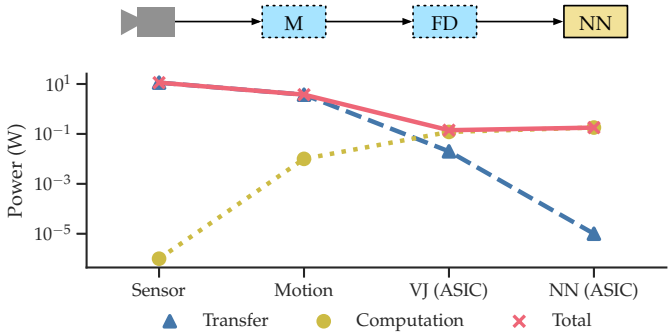


Figure 3.12: Computation and communication power after each ASIC block in the "full pipeline" configuration.

but not at the same rate. Counterintuitively, the total power increases by 28% after performing the NN, indicating that *it is more power-efficient to offload after motion and face detection, rather than performing the full pipeline.* Even though the NN only needs to communicate a 1-bit response, the cost of computation increases dramatically in comparison to the decrease in communication power. This result indicates that it is more cost-effective to offload the neural network than process it in-camera, in this power-constrained application domain.

So, why not always offload neural networks? We examined the extent to which different constraints in our design contributed to our results. We found that if the communication power cost per pixel grew by 2.68×, it would be more power-efficient to perform the NN in-camera. Our evaluation sourced communication power data from work using a directed RFID reader, and RF-based energy harvesting systems may not always deliver consistent power.

3.4.3 *Discussion*

Analysis of the video workloads highlights the benefits and limitations of our filtering approach at the application level. Importantly, the complete pipeline, consisting of motion detection, VJ, and NN, did not eliminate any true faces in any of our workloads. Using these filtering steps reduces data bandwidth, but we find that the extent of the data reduction could be improved. Motion detection often is triggered in innocuous situations, such as when people pass by in a frame or a mobile device is carried while walking. Our face detector misclassifies many false positives as faces, and also detects faces in static posters and photos. We examined the results of one of our security authentication workloads to illustrate: out of 62 frames of video, 12 frames were accepted through a motion detection block. In those 12 frames, the VJ detector passed forty 400-pixel face windows to the NN classifier. Visual inspection of those 40 faces found that 10% were false positives—with a perfect face detector that only detected true faces, the power to offload or compute the NN on-device would be reduced.

To investigate the impact of image size on the tradeoff between in-camera and offloading, we scaled our evaluation from the low-resolution images produced by the WISPCam to high-resolution mobile camera images. We found that keeping the neural network in-camera became power-efficient at 8 megapixels or greater. Finally, while offloading data may be more power-efficient, privacy and bandwidth concerns may lead designers to opt for a more localized solution when processing image data for face authentication tasks. These results indicate that as mobile camera devices become more sophisticated, architectures will have to become increasingly specialized to deliver low-power in-camera results.

3.5 SUMMARY

In this chapter, we use the notion of "in-camera processing pipelines" to thoroughly characterize the design of a highly constrained low-power camera system. Our results highlight how design parameters for individual accelerators can influence the full-system execution behavior, as well as shape decisions about whether to process a computation block in the camera or offload the computation. In particular, the inclusion of optional computational blocks—that are not essential for pipeline correctness, but increase overall computational efficiency by generating semantic perceptual information—can lead to significant cost reductions.

Power and performance constraints require increasingly efficient computational platforms, and architects will continue to look to hardware acceleration to enable challenging applications. In this section, we show how the benefits of accelerators can be a small fraction of the challenges of data communication in the system. Given the growth of image data production and requirements of modern vision algorithms, future computational camera applications require such a full system approach to maintain power and performance efficiency.

4

REAL-TIME VIRTUAL REALITY VIDEO CAPTURE

Virtual reality (VR) devices are becoming widely available, from camera rigs for video capture [9, 45], to headsets for immersive viewing [144, 165]. Real-time rendering of 3D-360° video can enable a wide range of VR applications, from live sports and concert broadcasting to telepresence. While the domains of VR video content capture and viewing are growing more popular, no system is capable of producing 3D-360° VR videos in real time as of yet. Camera rigs like Google Jump and Facebook Surround [9] allow users to capture standard 2D video and render it as 3D-360° videos. Real-time VR video capture, coupled with immersive headsets like Google Daydream, Oculus Rift, or Samsung Gear VR [65, 144, 165], can enable a wide range of applications, from live sports and concert broadcasting to tele-presence.

The Google Jump camera rig [9] is one example of a commodity VR video capture device, using 16 cameras to capture high-resolution (4K-1080p) overlapping video streams of a 360° scene. The collected video is used to estimate edge-aware flow fields, which are composited into a stereoscopic 3D-360° video. A key portion of this processing pipeline, flow estimation, is constructed around the *bilateral solver*, a fast and edge-aware algorithm that combines simple bilateral filters with domain-specific optimization problems [13]. This flow estimation algorithm consumes the majority of the processing time, despite using one of the fastest existing algorithms across thousands of cores [9].

In this chapter, we introduce a new algorithm for this flow estimation problem, the hardware-friendly bilateral solver (HFBS). HFBS achieves significant speedups over the bilateral solver with little accuracy loss. While Barron and Poole's bilateral solver [13] is challenging to parallelize on modern hardware, our "hardware-friendly" algorithm can be easily parallelized on GPUs and field-programmable gate arrays (FPGAs). To demonstrate, we design a scalable FPGA-based hardware accelerator for HFBS, employing specialized memory layout and reduced-precision fixed-point computation to achieve real-time results. Compared to the original bilateral solver, HFBS is 4× faster on a CPU, 32× faster on a GPU, and 50× faster on an FPGA. We evaluate the accuracy of HFBS on the depth superresolution task and show that our algorithm is faster than every more accurate algorithm, and more accurate than any faster algorithm.

This chapter makes two contributions: an algorithm for hardware-friendly bilateral solving, and a fixed-function FPGA accelerator implementing HFBS. To achieve fast performance while maintaining accuracy, we take a hardware-software codesign approach where both the algorithm and hardware substrate are developed in tandem. Our algorithm modifies the original bilateral solver to ensure memory access is predictable and therefore fast, and performs optimization using preconditioned gradient descent with momentum to reduce global communication and enable parallel execution. Our hardware accelerator explores fixed-point arithmetic and bilateral-grid-specialized memory layout to process large-resolution bilateral grids in a scalable way in real time.

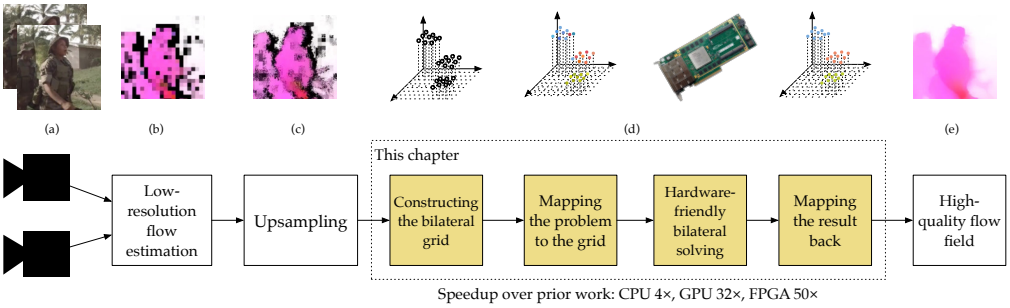


Figure 4.1: Our bilateral solver produces smooth, edge-aware flow fields. Given an input pair of images (a), a low-resolution flow is estimated (b), upsampled to a noisy high-resolution flow (c), and processed with the bilateral solver (d) to produce an edge-aware smoothed flow (e). Our algorithm for bilateral solving is better-suited for hardware acceleration and results in speedups of up to 50× over prior work [9, 13].

Many of these performance optimizations are codependent, and we evaluate performance of the algorithm and hardware together to illustrate our results. Our algorithm and accelerator design make it more practical to generate real-time VR video from camera rigs, either locally at the capture device, or in the cloud to accelerate large-scale video processing.

4.1 BACKGROUND: BILATERAL FILTERING AND THE BILATERAL GRID

We base our design for fast and accurate VR video on a state-of-the-art bilateral-space optimization algorithm, the bilateral solver [13]. The bilateral solver is general-purpose and scalable, and can be applied to the many vision applications: optical flow, stereo, depth superresolution, image colorization, and semantic segmentation. The bilateral solver can be used as part of an edge-aware optical flow algorithm for VR video, and scales to high resolutions efficiently [9].

This optical flow algorithm generates a correspondence map from a pair of images by computing a rough flow vector for every pixel (Figure 4.1b-c), and then refining that flow field until a cost function has been minimized. To compute this edge-aware per-pixel flow field, the bilateral solver resamples a coarse flow field into *bilateral-space* (Figure 4.1d), and then solves an optimization problem in bilateral-space to infer the smoothest possible flow-field that is similar to the input coarse flow field. In bilateral-space, simple local filters are equivalent to costly, global, edge-aware filters in pixel-space—consequently, flow refinement in bilateral-space is much faster than its pixel-space equivalent. We perform optimization in a three-dimensional *bilateral grid* data structure [30].

Figure 4.1 describes how we map our optical flow problem to the bilateral solver. We first compute a coarse, tile-based flow field, shown in Figure 4.1b, and then upsample it to noisy flow field. As seen in the figure, the upsampled flow in Figure 4.1c is very noisy, so we use the bilateral solver to smooth our noisy flow field while maintaining information . This last step (shown as a dotted box in Figure 4.1) translates into a computation-intensive optimization problem since multiple factors must be considered.

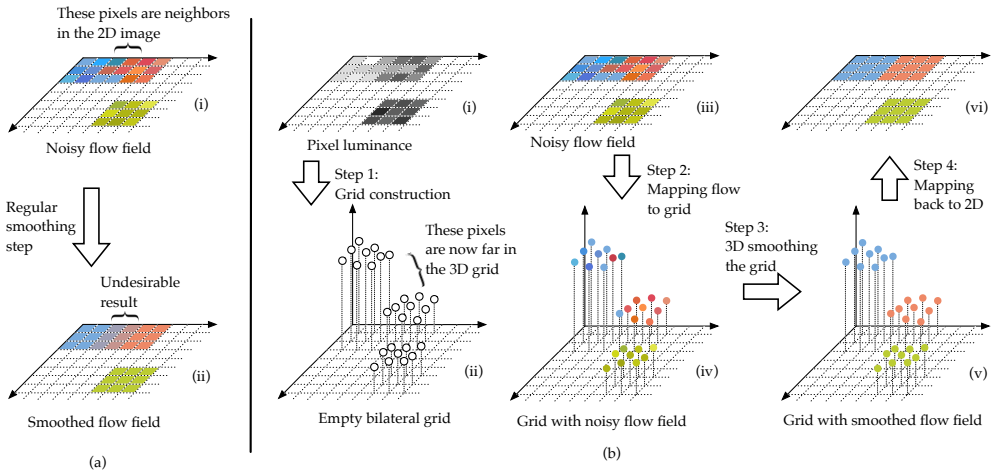


Figure 4.2: Smoothing a noisy flow field in (a) regular 2D space and (b) bilateral space. Regular smoothing produces undesirable artifacts at edges as the flow values blur together. The bilateral grid allows edge-aware smoothing and produces a correct denoised output.

To solve the optimization problem, the bilateral solver resamples the problem into *bilateral space*, which is a multi-dimensional space that allows edge-aware operations to be performed on an image. Figure 4.2 illustrates a simplified version of bilateral space and its use in our problem. We begin with the noisy flow field of Figure 4.2a-i, where color corresponds to some flow value. If we attempt to denoise this noisy flow field by applying a simple smoothing kernel, the result will present undesirable blurring at color edges. In Figure 4.2a-ii, for instance, the green region is successfully denoised, but the blue and red regions (which likely belong to different objects) blend around the edges, producing incorrect flow values there.

To smooth this flow field while maintaining sharp edges, we map the problem to bilateral space. First, we construct a *bilateral grid* for the original image, where a pixel in the image at location (x, y) with luminance l corresponds to a grid block at location (x, y, l) (Figure 4.2b-i, b-ii). In the 3D bilateral space, the lighter pixels are separated from neighboring darker pixels. We then map the flow value of each pixel (Figure 4.2b-iii) to its corresponding grid location (Figure 4.2b-iv). When we smooth this noisy flow in bilateral space, the blue and red areas are no longer neighbors and do not affect each other's value. Finally, we map the smoothed 3D flow (Figure 4.2b-v) back to the 2D representation (Figure 4.2b-vi). The resulting bilateral-smooth output in Figure 4.2a-vi retains sharp edges.

4.2 HARDWARE-FRIENDLY BILATERAL SOLVING

In this section, we formulate a bilateral solver that maintains speed, scalability, and accuracy, while also being parallelizable. We first describe the original bilateral solver of [13], and motivate the requirements for a hardware-friendly bilateral solver. We then provide a detailed formulation of our algorithm and its advantages.

4.2.1 *Bilateral-Space Optimization*

The original bilateral solver (OBS) consists of (1) an objective and (2) optimization technique [13]. The input to the solver is a reference RGB image, a target image that contains noisy observed quantities we wish to improve, and a confidence image. The goal is to recover an "output" vector $\mathbf{x}$, which will resemble the input target where the confidence is large while being smooth and tightly aligned to edges in the reference image. To achieve this, Barron and Poole construct an optimization problem of the following form:

$$\underset{\mathbf{x}}{\text{minimize}} \ \frac{\lambda}{2} \sum_{i,j} \hat{W}_{i,j} \left(x_i - x_j\right)^2 + \sum_i c_i (x_i - t_i)^2 \tag{4.1}$$

The first term of the loss encourages that for all pixel pairs i and j, the overall difference between their flow values x_i and x_j is minimized if they are neighboring pixels in the bilateral space. The second term of Eq.4.1 encourages each pixel x_i to be close to the target input t_i if that pixel's confidence c_i is high.

The affinity matrix $\hat{\mathbf{W}}$ is a *bistochastized* (all rows and columns sum to 1) version of a bilateral affinity matrix $\mathbf{W}$. Each element of the bilateral affinity matrix $W_{i,j}$ describes the affinity between pixels i and j in the reference image in the YUV colorspace:

$$W_{i,j} = \exp\left(-\frac{\left\| \left[p_i^x, p_i^y\right] - \left[p_j^x, p_j^y\right] \right\|^2}{2\sigma_{xy}^2} - \frac{\left(p_i^l - p_j^l\right)^2}{2\sigma_l^2} - \frac{\left\| \left[p_i^u, p_i^v\right] - \left[p_j^u, p_j^v\right] \right\|^2}{2\sigma_{uv}^2} \right) \tag{4.2}$$

where p_i is a pixel in the reference image with location (p_i^x, p_i^y) and color (p_i^l, p_i^u, p_i^v). The σ_{xy}, σ_l, and σ_{uv} parameters control the support of the spatial, luminance (luma), and chrominance (chroma) components of the filter. Bistochastization normalizes this affinity matrix while maintaining symmetry [12].

Bilateral operations (e.g., filtering) can be sped up by treating the filter as a "splat/blur/slice" procedure in the bilateral grid. The splat/blur/slice filtering approach corresponds to a compact factorization of $\mathbf{W}$:

$$\mathbf{W} = \mathbf{S}^{\mathrm{T}} \mathbf{B} \mathbf{S} \tag{4.3}$$

where $\mathbf{S}$ and $\mathbf{S}^{\mathrm{T}}$ are splatting and slicing, and $\mathbf{B}$ is a [1 2 1] blur kernel. As in [13], $\mathbf{S}$ defines a per-pixel mapping from a pixel to a coarse bin in the bilateral grid, where that mapping is a function of the x and y coordinates, l luma, u and v chroma of that pixel. Multiplying by $\mathbf{S}$ is a data-dependent histogramming operation, and multiplying by $\mathbf{S}^{\mathrm{T}}$ is a data-dependent interpolation. The bilateral space optimization formulation of [12] performs bistochastization by calculating two matrices $\mathbf{m}$ and $\mathbf{n}$ that satisfy the following:

$$\hat{\mathbf{W}} = \mathbf{S}^{\mathrm{T}} \operatorname{diag}\left(\frac{\mathbf{n}}{\mathbf{m}}\right) \mathbf{B} \operatorname{diag}\left(\frac{\mathbf{n}}{\mathbf{m}}\right) \mathbf{S} \tag{4.4}$$

where $\hat{\mathbf{W}}$ is a bistochastic version of matrix $\mathbf{W}$. The vectors $\mathbf{m}$ and $\mathbf{n}$ describe a normalizing transformation required by the solver.

Barron and Poole also perform a variable substitution [13], transforming the high-dimensional pixel-space optimization problem into one with lower-dimensional bilateral-space vertices:

$$\mathbf{x} = \mathbf{S}^{\mathrm{T}}\mathbf{y} \tag{4.5}$$

where $\mathbf{y}$ is a small vector of values for each bilateral grid vertex, and $\mathbf{x}$ is the large vector of values for each pixel.

Equations 4.3 and 4.5 allow us to reformulate the pixel-space loss function of Eq. 4.1 into bilateral-space in a quadratic form:

$$\begin{aligned} &\underset{\mathbf{y}}{\text{minimize}} \quad \frac{1}{2}\mathbf{y}^{\mathrm{T}}\mathbf{A}\mathbf{y} - \mathbf{b}^{\mathrm{T}}\mathbf{y} + c \\ &\mathbf{A} = \lambda(\operatorname{diag}(\mathbf{m}) - \operatorname{diag}(\mathbf{n})\mathbf{B}\operatorname{diag}(\mathbf{n})) + \operatorname{diag}(\mathbf{S}\mathbf{c}) \\ &\mathbf{b} = \mathbf{S}(\mathbf{c}\circ\mathbf{t}) \qquad c = \frac{1}{2}(\mathbf{c}\circ\mathbf{t})^{\mathrm{T}}\mathbf{t} \end{aligned} \tag{4.6}$$

where $\mathbf{y}$ is the solution to the problem in bilateral-space, $\mathbf{m}$ and $\mathbf{n}$ are defined by Eq. 4.4, and $\mathbf{t}$ and $\mathbf{c}$ are per pixel initial solutions and confidences (Eq. 4.1). The Hadamard (element-wise) product is denoted by $\circ$.

The optimization problem of Eq. 4.1 is intractably slow to solve naively. However, the bilateral-space formulation allows feasible and fast execution. Minimizing Eq. 4.6 is equivalent to solving a sparse linear system:

$$\mathbf{A}\mathbf{y} = \mathbf{b}$$

and we can produce a pixel-space solution $\hat{\mathbf{x}}$ by slicing out the solution from the linear system:

$$\hat{\mathbf{x}} = \mathbf{S}^{\mathrm{T}}(\mathbf{A}^{-1}\mathbf{b}) \tag{4.7}$$

In summary, OBS takes an input image vector and a confidence image to construct a simplified bilateral grid from the reference image. With that, it produces the $\mathbf{A}$ matrix and $\mathbf{b}$ vector of Eq. 4.6 to solve the linear system in Eq. 4.7 and obtain an output image.

4.2.2 *Algorithmic Modifications*

Though computationally efficient, OBS as presented has a number of properties that make it difficult to implement in hardware, or even to achieve real-time operation on modern CPU or GPU systems. Vectorizing and parallelizing CPU or GPU processing on the sparse 5D bilateral grid $\hat{\mathbf{W}}$ demonstrates too-irregular memory access patterns to achieve large performance benefits from parallelization. Moreover, the use of second order global optimization limits the level of parallelism we can extract from the algorithm. We modify OBS to construct a hardware-friendly bilateral solver and address these specific challenges: color and sparse memory indexing, and second order global optimization. Our modifications also allow for an alternative, more efficient initialization and reduced quantization artifacts, which we will discuss after formulating our algorithm.

COLOR AND SPARSE MEMORY INDEXING. The bilateral solver of [13] was designed around a hard bilateral grid or a permutohedral lattice [3], meaning that optimization takes place in a "sparse" five-dimensional bilateral space (where the five dimensions are position in x and y, pixel luma, and two pixel chroma values). The resulting 5D grid has an image-dependent "sparsity" that is challenging to exploit in parallel algorithms. Moreover, the connectivity structure of the graph used in the bilateral solver varies as a function of the input, leading to expensive and unpredictable memory access patterns. Attempting to resolve this by solely converting the sparse grid into a "dense" representation of the 5D space requires a prohibitive amount of memory. Instead, HFBS ignores the color of the input image and uses a "dense" 3D bilateral grid [30], which makes memory indexing predictable and enables further optimizations. Ignoring color this way induces a small decrease in accuracy, as we will demonstrate.

SECOND ORDER GLOBAL OPTIMIZATION. The numerical optimization in OBS was performed using the preconditioned conjugate gradient method with a Jacobi or Jacobi-like hierarchical preconditioner. Conjugate gradient methods use a global optimization step: at each iteration, updating each variable of the optimization vector requires reasoning about the gradient at all other variables. Such global communication requirements make parallel hardware implementation difficult, as we want to be able to individually update and optimize any variable in our state space via local communication with the "neighboring" variables in our bilateral grid. To avoid global communication, HFBS performs optimization using gradient descent with momentum (i.e., the "Heavy Ball" algorithm), which can be shown to have similar asymptotic performance as conjugate gradient [152]. This converts an irregular number of global matrix operations into a regular, but larger, number of local updates that are much easier to execute in parallel.

The Heavy Ball algorithm does not naturally accommodate a preconditioner, so we reformulate our optimization problem with a transformation that indirectly applies a Jacobi preconditioner during optimization. We find that HFBS slightly underperforms the preconditioned conjugate gradient solver of [13] and therefore requires roughly twice as many steps for convergence. However, since each step is significantly faster to compute (roughly 4× faster on CPU and much faster on GPU/FPGA), we see an overall increase in performance.

4.2.3 *Formulating a Hardware-friendly bilateral solver*

We now formalize the details of HFBS and how it relates to the original bilateral solver. Both OBS and HFBS minimize an optimization problem of the form of Eq. 4.1. In this case, the t_i is the low resolution flow shown in Figure 4.1b. We derive the confidence image for these low resolution flow fields by computing normalized sum-of-squared differences. The resulting confidence is larger for areas that are near each other and match well.

We obtain the weight $\hat{W}_{i,j}$, which determines the bilateral-space distance between two pixels i and j, from a bistochastized version of the matrix $\mathbf{W}$ whose elements are calculated via the following:

$$W_{i,j} = \exp\left(-\frac{\left\|[[p_i^x, p_i^y] - [p_j^x, p_j^y]\right\|^2}{2\sigma_{xy}^2} - \frac{(p_i^l - p_j^l)^2}{2\sigma_l^2}\right) \tag{4.8}$$

where each pixel p_i has a spatial position (p_i^x, p_i^y) and luminance p_i^l. While OBS includes color information in $\hat{W}_{i,j}$ (Eq. 4.2), HFBS only considers luminance.

The bistochastization step in [13] requires 10-20 iterations to achieve low error. To reduce the fixed cost of this step, we use a faster, approximate bistochastization step for initializing the bilateral solver. Unlike OBS, which fully-bistochastizes $\mathbf{W}$ into $\hat{\mathbf{W}}$, we construct an approximately bistochastized $\hat{\mathbf{W}}$ (equivalent to one iteration of bistochastization) that still satisfies the requirements of the bilateral solver:

$$\mathbf{m}_0 = \mathbf{S1} \qquad \mathbf{n} = \sqrt{\frac{\epsilon + \mathbf{m}_0}{\epsilon + \mathbf{B1}}} \qquad \mathbf{m}_1 = \mathbf{n} \circ (\mathbf{Bn}) \tag{4.9}$$

In OBS, bistochastization is done to convergence, which produces a $\mathbf{n}$ which satisfies $\mathbf{m}_0 = \mathbf{n} \circ (\mathbf{Bn})$. Partial bistochastization requires that we treat this equality as an assignment, thereby constructing $\mathbf{m}_1$ to explicitly obey this constraint (Eq. 4.9). This produces nearly-indistinguishable output while being faster and easier to compute.

Our normalization also differs from OBS by the use of $\epsilon \sim 0.00001$ in the construction of $\mathbf{n}$. Adding ϵ to the numerator prevents divide-by-zero later and ensures that empty grid cells do not propagate information during optimization. Adding it to the denominator prevents the addition of ϵ in the numerator from biasing the solution towards 0. Note that the partial bistochastization step of HFBS is not iterative and does not require any convergence, and thus is significantly faster than the bistochastization step of OBS.

As described earlier, the expensive per-pixel optimization in Eq. 4.1 can be reformulated to a much more tractable optimization problem inside a bilateral grid. For convenience we will define $\mathbf{By}$ (the product of some grid $\mathbf{y}$ with a blur $\mathbf{B}$) as a scaling of and "diffusion" of $\mathbf{y}$:

$$\begin{aligned} \mathbf{By} &= 2\mathbf{y} + \mathbf{Dy} \\ \mathbf{Dy} = D(\mathbf{y}) &= \mathbf{y}(x+1, y, z) + \mathbf{y}(x-1, y, z) \\ &\quad + \mathbf{y}(x, y+1, z) + \mathbf{y}(x, y-1, z) \\ &\quad + \mathbf{y}(x, y, z+1) + \mathbf{y}(x, y, z-1) \end{aligned}$$

where $\mathbf{D}$ is a diffusion operator (which we can interchangeably refer to as a matrix and a function) that replaces each element in $\mathbf{y}$ with the sum of its neighbors. Because our 3D bilateral grid is dense in memory, this diffusion process is a simple stencil operation.

We now perform a variable substitution, as in Eq. 4.5. For us, this simply requires dividing by the square root of the diagonal of the $\mathbf{A}$ matrix:

$$\mathbf{y} = \mathbf{p} \circ \hat{\mathbf{z}} \qquad \mathbf{p} = \frac{1}{\sqrt{\mathbf{Sc} + \lambda\,(\mathbf{m}_1 - 2\,(\mathbf{n} \circ \mathbf{n}))}}$$

where $\hat{\mathbf{z}}$ is the solution to the substituted problem.

With our variable substitution in place, we can reformulate Eq. 4.6:

$$\begin{aligned}
\underset{\mathbf{z}}{\text{minimize}} \quad & \frac{1}{2}\mathbf{z}^{\mathrm{T}}\tilde{\mathbf{A}}\mathbf{z} - \tilde{\mathbf{b}}^{\mathrm{T}}\mathbf{z} + c \\
\tilde{\mathbf{A}} &= \mathbf{I} - \operatorname{diag}(\mathbf{q})\,\mathbf{D}\operatorname{diag}(\mathbf{q}) \\
\tilde{\mathbf{b}} &= \mathbf{p} \circ (\mathbf{S}(\mathbf{c} \circ \mathbf{t})) \\
\mathbf{q} &= \sqrt{\lambda}\,(\mathbf{n} \circ \mathbf{p})
\end{aligned}$$

Here c is the same as in Eq. 4.6. Note that the diagonal of $\tilde{\mathbf{A}}$ is $\mathbf{1}$, so optimizing this problem without a preconditioner is the same as optimizing Eq. 4.6 with a Jacobi preconditioner. Minimizing this problem requires solving a linear system, undoing our preconditioning variable substitution, and then slicing out a solution:

$$\hat{\mathbf{z}} = \tilde{\mathbf{A}}^{-1}\tilde{\mathbf{b}} \qquad\qquad \hat{\mathbf{x}} = \mathbf{S}^{\mathrm{T}}(\mathbf{p} \circ \hat{\mathbf{z}})$$

We will solve this problem using the "Heavy Ball" method (gradient descent with momentum). This problem is fully-described by the diffusion operator $D(\cdot)$ and the bilateral grids $\tilde{\mathbf{b}}$ and $\mathbf{q}$.

Algorithm 1 shows pseudocode describing how optimization is performed:

Algorithm 1 Bilateral-Space Heavy Ball Method

Input: problem description $\{D(\cdot), \tilde{\mathbf{b}}, \mathbf{q}\}$, initial state $\mathbf{z}_{\text{init}}$, step size $\alpha = 1$, momentum $\beta = 0.9$, number of iterations $n = 256$.

Output: state after n iterations $\mathbf{z}$.

$\mathbf{z} \leftarrow \mathbf{z}_{\text{init}}$
$\mathbf{h} \leftarrow \mathbf{0}$
for $i = 1 : n$ **do**
    $\mathbf{g} \leftarrow \mathbf{z} - \mathbf{q} \circ D(\mathbf{q} \circ \mathbf{z}) - \tilde{\mathbf{b}}$
    $\mathbf{h} \leftarrow \beta\mathbf{h} + \mathbf{g}$
    $\mathbf{z} \leftarrow \mathbf{z} - \alpha\mathbf{h}$
end for

It can be shown that if the momentum and step size hyperparameters are set correctly, this heavy ball method has the same asymptotic performance as conjugate gradient [152]. Because preconditioning has been absorbed into the problem, performance approaches preconditioned conjugate gradient. Since the diffusion operator $D(\cdot)$ is a local stencil, the gradient update to $\mathbf{g}$ and the optimization update to $\mathbf{h}$ and $\mathbf{z}$ can be performed efficiently (i.e., vectorized, parallelized, etc).

BETTER INITIALIZATION TO REDUCE OPTIMIZATION ITERATIONS Our objective function is convex and thus invariant to the initialization $\mathbf{z}_{\text{init}}$, but a better

initialization may allow us to converge in fewer iterations. We can achieve this with a simple weighted blur in our bilateral grid.

$$\mathbf{z}_{\text{init}} = \frac{\text{blur}(\mathbf{S}(\mathbf{c}\circ\mathbf{t}), \sigma_b)}{\mathbf{p}\circ \text{blur}(\mathbf{S}(\mathbf{c}), \sigma_b)}$$

where $\text{blur}(\mathbf{a}, \sigma_b)$ is a large-support 3D Laplacian blur of $\mathbf{a}$ with a scale of σ_b:

$$\text{blur}(\mathbf{a}, \sigma_b)(t_x, t_y, t_z) = \iiint_{-\infty}^{\infty} e^{\left(\frac{-|\tau_x|-|\tau_y|-|\tau_z|}{\sigma_b}\right)} \mathbf{a}(t_x - \tau_x, t_y - \tau_y, t_z - \tau_z) d\tau_x d\tau_y d\tau_z$$

and t_x, t_y, and t_z are 3D coordinates. This can be efficiently implemented as three separable infinite impulse response filters (i.e., exponential smoothing, forward and backward) in the three dimensions of the grid. The intuition behind this initialization is that the solution should be close to $\mathbf{b}$ where the confidence is large and smooth where confidence is small. We found that this initialization can be implemented efficiently on a CPU and roughly halves the number of required iterations.

REDUCED QUANTIZATION ARTIFACTS In OBS, slicing can introduce "blocky" quantization artifacts in the output [13]. This quantization requires post-processing, adding to the computational cost of running the bilateral solver. However, HFBS uses a dense and low-dimensional grayscale bilateral grid which allows us to efficiently slice out of our bilateral grid using trilinear interpolation. As shown in [30], this produces smooth results without post-processing. The trilinear interpolation can be done through a weighted slice, where $\mathbf{S}_{\text{tri}}$ is analogous to $\mathbf{S}$ but with trilinear weights instead of hard "one-hot" assignment:

$$\hat{\mathbf{x}} = \frac{\mathbf{S}_{\text{tri}}^{\text{T}}(\mathbf{m}_0 \circ \mathbf{p} \circ \hat{\mathbf{z}})}{\mathbf{S}_{\text{tri}}^{\text{T}}(\mathbf{m}_0)}$$

By performing a weighted slice according to the per-vertex grid occupancy $\mathbf{m}_0$ this process produces artifact-free results in comparison to results from OBS, even if trilinear interpolation is not used in the splatting step. This "soft" slicing is only slightly more expensive than its "hard" equivalent, though both forms can be implemented very efficiently by virtue of being simple gather operations in a dense bilateral grid.

4.3 HARDWARE ARCHITECTURE

The formulation of HFBS allows for fast bilateral solving on high-performance CPUs or GPUs, but the resulting power consumption may prove prohibitively costly for a full system. FPGA platforms, on the other hand, can demonstrate fast performance with better power efficiency. This makes them a more suitable target for a system requiring multiple high-performance processors in a single chassis that can support processing 16-camera outputs simultaneously. To demonstrate power and performance efficiency on FPGAs, we co-designed our hardware implementation with the HFBS algorithm. In addition to the algorithmic optimizations, we apply hardware-specific techniques such

as customized variable bitwidths and bilateral-space memory partitioning to enable better performance.

We first discuss the hardware system at a high level, and then our specific design exploration for bitwidth precision and bilateral grid memory layout. Finally, we describe the hardware-software interface of our design and how we integrate the accelerator into an application.

4.3.1 *Microarchitectural Design*

We focus on executing the inner loop of Algorithm 1 with custom hardware, and maintaining the higher-level control flow in software. In this scheme, a software application splats the optimization problem defined in Section 4.2 onto a bilateral grid, and transfers it to the accelerator for iterative solving. Figure 4.3 shows a high-level overview of how application functionality is distributed across the system. The figure also illustrates details of our design's microarchitecture.

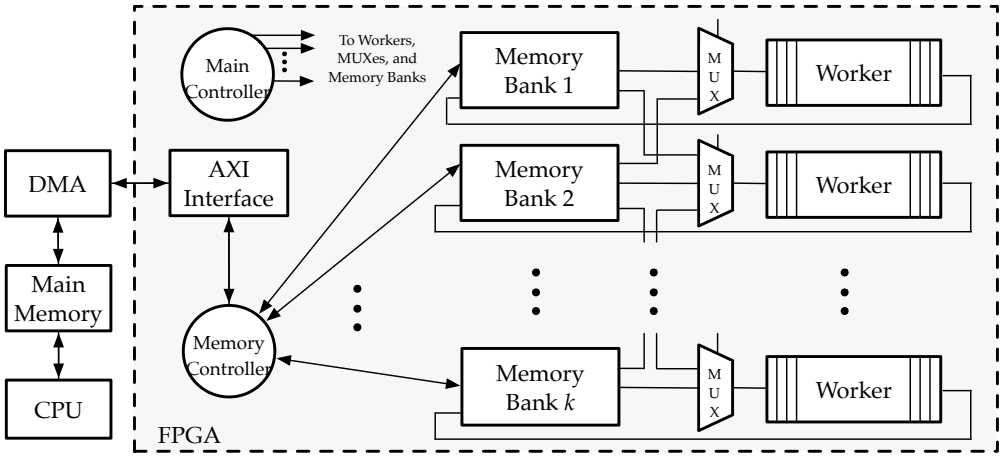


Figure 4.3: High-level system overview of our accelerator. Parallel workers process bilateral grid vertices stored in partitioned memory banks. Neighboring bank access is controlled with MUXes.

DATA TRANSFER PROCESS. The CPU constructs the bilateral grid based on the input reference image and the initial low-resolution solution provided from prior steps. This data is splatted onto the grid and transferred to the accelerator via direct memory access (DMA). The transferred data includes $\hat{\mathbf{b}}$, $\mathbf{q}$, and the initial solution $\mathbf{z}_{\text{init}}$ shown in Algorithm 1.

First-in-first-out queues (FIFOs) buffer data transfers at the input and output of the accelerator. During transfer, the memory controller of Figure 4.3 interleaves the data corresponding to each bilateral grid vertex, and partitions the data into memory banks. After the data transfer is complete, a pool of parallel workers iteratively solve the optimization problem by running the loop of Algorithm 1. Multiple grid vertices

are processed in parallel and the results are updated in place. After some number of iterations (we chose 256 iterations for our experiments to ensure convergence), the CPU reads back the final solution and slices it into a 2D result.

Each worker (shown in in Figure 4.4) performs the inner loop of Algorithm 1 on one grid cell.

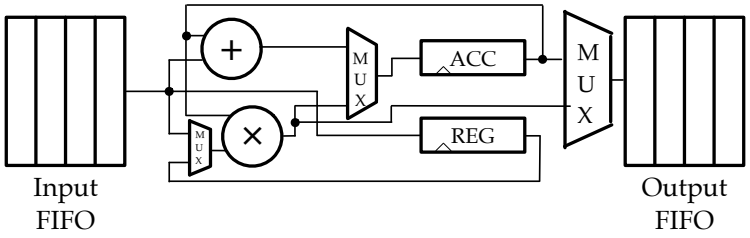


Figure 4.4: Block diagram of a single worker in our design. Each worker receives the data for a grid vertex's stencil computation through an input FIFO, processes the grid vertex, and then sends the result out through an output FIFO.

It computes the result by streaming in the data from the neighboring cells (required for the "blur" operation), as well as the normalization factors required for the optimization process. For each grid cell, a total of 6 add/subtract operations and 3 multiplications are performed. The updated result is written back to the memory. The workers compute their local stencil operations synchronously, interfacing primarily with an assigned memory bank and occasionally the neighboring memory banks to access grid blocks that may be stored across banks. Because each worker executes in lockstep, there are no memory collisions when accessing data in other banks. Figure 4.3 demonstrates how multiplexers, managed by the main controller, shepherd access to neighboring banks. As we scale the number of workers, we find that parallelism introduces a 1% reduction in speedup against perfect linear scaling. This near-linear scaling can entirely be attributed to our inclusion of the "Heavy Ball" algorithm in HFBS, which allowed our design to use only local-neighbor communication rather than global synchronization after each iteration.

Ideally, we want the memory to be partitioned along one grid dimension only, to simplify the neighbor connections between the workers (i.e., simplify the MUXes in Figure 4.3). However, this limits the number of parallel workers, to the number of grid blocks along the selected dimension. If more parallel workers are necessary, the memory should be partition along multiple dimensions, leading to a more complicated neighbor connections.

NUMBER OF WORKERS The number of workers is affected by several parameters such as run time requirements, power budget, and available resources. Figure 4.5 shows how performance scales with the number of workers. Assuming we want the fastest design and have not power constraint, the number of workers will be determined by the available FPGA resources including digital signal processing units (DSPs), look-up tables (LUTs), and memory banks.

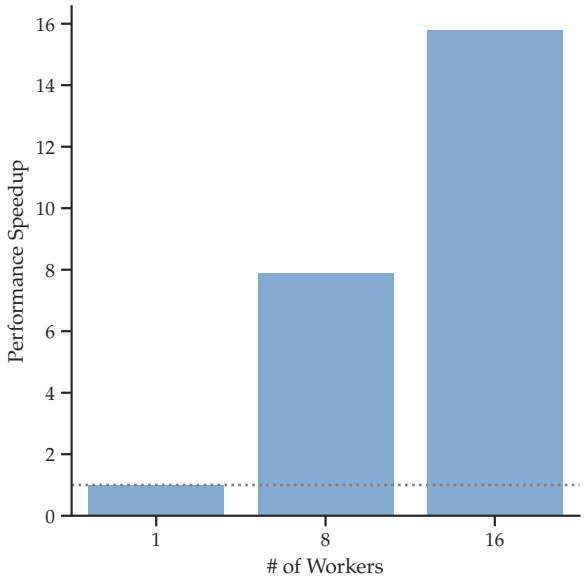


Figure 4.5: Performance scales almost linearly with increased numbers of workers.

FIXED-POINT CONVERSION To improve resource utilization, we converted the algorithm from floating-point to fixed-point number representation. We first implemented our workers using single-precision floating-point, like our CPU and GPU implementations. We found that the large number of digital signal processing units (DSPs) required for a single floating-point multiplier prohibitively limited the number of workers we could employ, and consequently, the amount of parallelism. Converting FPGA designs from floating-point to fixed-point number representation resolves this by reducing the resource requirements of hardware multipliers. Using the cheaper fixed-point multiplier, however, required us to evaluate three competing tradeoffs: (1) the bitwidth of our fixed-point numbers, (2) the precision at a given bitwidth for the integer and fractional portions of the number, and (3) convergence of the solver. If less than 12 bits were used for the integer portion, the bilateral grid data would quickly populate with overflow values. If less than 24 bits were used for the fractional component of the number, the bilateral solver would not converge, because grid vertices would not have enough precision to capture the change in a value after blurring.

These constraints prevented us from using 32-bit fixed-point numbers, as highlighted by the high mean squared error (MSE) shown in Figure 4.6 across integer-fraction-ratio configurations. We delineate a maximum error threshold of ~ 0.00001, because any errors exceeding that precision eliminate the positive benefits of using an ϵ-value to reduce zero-propagation. Qualitatively, we also observed that a low MSE correlated with visually similar solver output that converged at the same rate as a 32-bit floating-point solver. Using 64-bit fixed-point numbers resulted in very low MSE, but, as seen in Table 4.1, required 16 DSPs per worker, limiting the number of parallel workers we could deploy with these configurations.

As a compromise, we evaluated a 47-bit number representation that was more accurate than 32-bit fixed-point, with 75% less DSPs than 64-bit fixed-point. To maintain

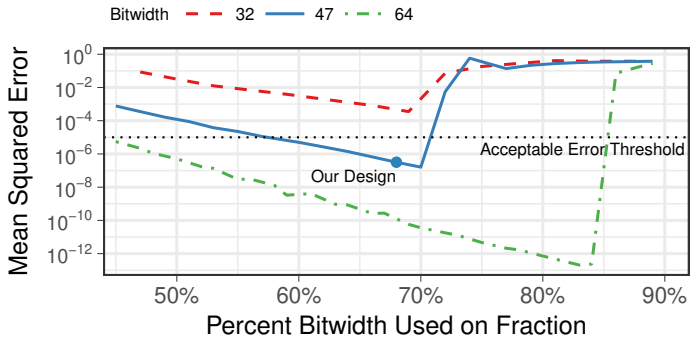


Figure 4.6: MSE of fixed-point implementations at varied fractional widths, for different bitwidths. MSE is reported relative to 32-bit floating-point. We chose a configuration with 31 bits of fractional precision to reduce chance of overflow in the integer portion.

Table 4.1: Worker resource utilization and maximum workers at varied bitwidths. Reported MSE is relative to 32-bit floating-point.

Bitwidth	32	47	64
DSPs per Worker	1	4	16
Maximum Workers	6840	1710	427
Min. MSE	8.30×10^{-4}	6.69×10^{-7}	7.16×10^{-13}

some precision of 64-bit numbers during non-multiplier arithmetic, we chose a 64-bit fixed-point representation with 15 bits of integer precision and 48 bits of fractional precision, and cast it to and from 47-bit for multiply operations only. Before multiplying two 64-bit numbers, we round off the bottom 16-bits of each number, resulting in the 1-bit sign, 15-bit integer, 31-bit fraction number highlighted in Figure 4.6. We zero-extend the resulting 47-bit output back to a 64-bit number for the rest of the computation. This fixed-point configuration has a MSE of 3.17×10^{-7} compared to the floating-point implementation, resulting in negligible accuracy loss at the solver output, and the solver converges at the same number of iterations.

4.3.2 *Bilateral Grid Storage and Memory Layout*

GRID STORAGE SIZE'S IMPACT ON QUALITY. Instead of a simple filter like moving average, BSSA maps a noisy depth map to a bilateral grid, refines the depth map by solving an optimization problem, and remaps the bilateral-grid result to pixel-space. Varying the number of pixels that map to a grid vertex impacts the time to compute the stereo refinement for a frame, and also the quality of the depth map. Figure 4.7 demonstrates the tradeoff between stereo image quality and bilateral grid size to be processed for high-resolution input images.

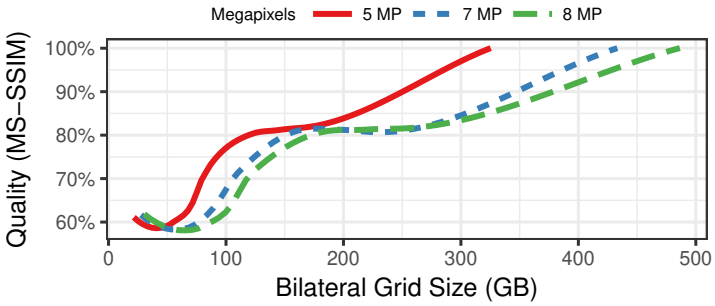


Figure 4.7: Using a smaller bilateral grid is cheaper to compute but degrades the quality of the output depth map, even at high image resolutions.

Here, we scaled bilateral grid sizes from 4 pixels-per-grid-vertex to 64 in each of three dimensions in a bilateral grid and evaluated the resulting impact on quality using MS-SSIM [186]. We find the resolution of the input images is less impactful than choosing an appropriate grid size to balance quality and computational complexity.

ON-CHIP MEMORY LAYOUT. To take advantage of block RAM distribution on the FPGA, we partitioned the bilateral grid into chunks along different dimensions, and dedicated grid workers for each partition. For large, finely divided grids with many vertices (the largest grids we consider have up to 5 million vertices), we could achieve full resource utilization simply by partitioning the grid along one dimension and allocating a single worker to process each memory bank. For more coarse grids, we partitioned the memory in multiple dimensions.

Our method for laying out data in memory consists of storing all the data needed for a grid vertex in a single packet, and writing the packets sequentially in memory. Rather than storing multiple bilateral grid data structures separately and repeatedly indexing into each of them to process a single vertex, we interleave the data structures together to access all the information for processing a grid vertex as a single packet. When a worker is assigned a grid vertex to process, it can fetch most of the data required for its computation from a single partition, including neighboring vertex data for some dimensions. For large grids, where we only partition on one dimension, the data for two of the three dimensions is stored in the same memory bank, and the worker only has to communicate across banks for the two neighbors in other partitions. For smaller grids, where we partition along multiple dimensions to improve parallelism, workers may need to fetch more of their neighbors from neighboring partitions. All inter-bank communication is handled via the main controller of Figure 4.3.

To aid in fetching grid vertex data for a worker's vertex or neighboring ones, we abstracted this memory layout into a simple addressing method: we dedicate $\lceil \log_2(k) \rceil$ bits of address space for each grid dimension with size k, and use the last three bits to index into the packet for a grid vertex. For instance, with a bilateral grid of shape $\lfloor 247, 166, 16 \rfloor$ partitioned on the first dimension only, a worker assigned the address `0b 00001010 10100001 0100 001` would map the first dimension's value to memory bank 10, and use the second and third dimensions to fetch the second item in the packet

for grid vertex $[10, 161, 4]$. Indexing into a neighboring vertex in any dimension means incrementing or decrementing a dimension's tag; the main controller detects when a worker is requesting an address in a neighboring bank and multiplexes the request appropriately. This discrete mapping of grid dimensions to address spaces results in simple logic for memory addressing, but at the cost of wasted memory space. Each grid vertex packet contains five items but requires the memory space for eight. The same is true at the grid partition level, since the number of grid vertices along a dimension is a function of the image resolution and the σ_{xy} or σ_l, and does not often fit nicely in power-of-two partitions.

4.3.3 *FPGA Implementation*

To invoke the bilateral solver accelerator in an application, the application sends a bytestream of bilateral grid data over the FPGA's PCIe-to-AXI DMA interface. Once the accelerator finishes processing the bilateral grid, it sends the data back in a bytestream to the program over the same interface. The FPGA's driver can be invoked with standard Unix I/O system calls like $read()$ and $write()$, and can thus be integrated with software applications written in any high-level language.

Some parameters of our design are fixed at configuration time, while others can be modified by software. At configuration time, we fix the number of grid workers, memory size, and partitioning based on a chosen set of parameters for image resolution and bandwidths in the luma and spatial dimensions. The parameters essentially define the maximum memory size and number of partitions, which can be interpreted as the upper bound of grid sizes that can be run under a certain configuration. The bilateral grid dimensions and number of iterations are software-defined at program runtime. Before an application sends a bilateral grid to the accelerator for processing, it first sends these software-defined parameters. If the program requests to process a grid too large for the FPGA's configuration, the accelerator will return an error. This level of flexibility in our design allows applications to process images of varied resolutions at varied grid bandwidths, but can result in wasted resources if the grid size being processed is much smaller than the accelerator's configured grid size.

4.4 EXPERIMENTAL METHODOLOGY & RESULTS

We designed HFBS with the goal of improving bilateral solver performance by parallelizing on modern hardware, while maintaining comparable visual accuracy. We compare our algorithm across CPU, GPU, and FPGA implementations. The CPU is a Xeon E5-2620 with six cores, and the GPU is an NVIDIA GTX 1080 Ti. Both platforms execute optimized implementations of the kernels written and tuned with Halide [157]. We prototyped our FPGA design on two devices, and simulated a third. To evaluate performance and host-device memory traffic, we experimentally prototyped on a Xilinx Zynq-7020 SoC and a Xilinx Kintex-7 connected to a host CPU over PCIe. We also synthesized and simulated a target evaluation design for the Xilinx Virtex UltraScale+

VU9P to evaluate full-resolution frame processing. Details of each platform are listed in Table 4.2.

Table 4.2: Evaluated platforms

Platform	Type	Cores	Process (nm)	Freq (MHz)
Xeon E2620	CPU	6	32	2000
Nvidia 1080 TI	GPU	3584	16	1582
Zynq-700	FPGA	N/A	28	125
Kintex Ultrascale+	FPGA	N/A	16	250

Both FPGA-based accelerator systems use DMA, either over an on-chip interconnect for the Zynq or PCIe for the Kintex devices. The CPU prepares the bilateral-grid data structure with pixels mapped to grid vertices, and transfers them via DMA to the FPGA fabric. The hardware accelerator processes the vertices with the bilateral-space filtering and streams them back to the CPU, where the bilateral-grid-filtered result is converted into the fully-processed depth map.

Applying the computation of B_3 to a high-resolution video is equivalent to applying millions of blurs to the bilateral grid representation of the video frames. Across a single frame, most of these filters can run in parallel, so we designed streaming compute units to run bilateral filters on a stream of grid vertices. We find that BSSA requires at least 32-bit floating-point precision to produce high-quality depth maps, and use DSP units on the FPGA fabric to compute efficient floating-point operations. Each compute unit requires 18 DSP units in our design, so we can scale up to 12 parallel compute units on the ZC702. However, we project that if we scale up to a top-of-the-line Xilinx Virtex UltraScale+ FPGA, we can parallelize up to 682 compute units, which are more than enough for real-time operation. Table 4.3 summarizes the setup we use in our evaluation and resource requirements for real-time performance with a 16-camera system.

Table 4.3: Requirements for FPGA acceleration platform.

	Resource	Evaluation	Target
System	FPGA Model	Zynq-7000	Virtex UltraScale+
	FPGA (#)	1	16
	Cameras	2	16
Per FPGA	Logic	46%	44%
	RAM	7%	99%
	DSP	94%	100%
	Clock (MHz)	125	250

BENCHMARKING. To benchmark our algorithm, we execute the bilateral solver on flow fields and confidences generated from the ten training images in the Middlebury stereo dataset [167], and evaluate runtime and accuracy. We compare runtimes for our algorithm on CPU, GPU, and FPGA with the bilateral solver of [13] on CPU as the baseline. For CPU and GPU implementations, we report the average runtime from 8 trials; the FPGA runtime is deterministic and did not vary across trials. We characterize power consumption for the CPU and GPU by measuring utilization and scaling from the reported device power. For the FPGA, we report estimated power consumption from Xilinx Vivado's power report.

BILATERAL GRID SIZES. The size of the bilateral grid data ranges from 4KB-1.8GB, depending on the σ_{xy} used to construct the grid. All results use a $\sigma_l = 16$. We use 256 iterations of optimization in all cases, more than enough guarantee convergence for all algorithms and implementations. Note that our performance comparison is not at iso-quality, as our algorithm has slightly more error but qualitatively similar results, which we discuss more in Section 4.4.1. All computation is single-precision floating-point, except for bilateral solving on the FPGA which is conducted with 64-bit fixed-point numbers. We observe transfer throughput for the FPGA over a single PCIe channel to range between 9.6-11.3 Gbps, which is in keeping with reported estimates. Since both GPU and FPGA communicate with the host over PCIe and we assume frames can be pipelined, we omit the transfer time between the host processor and the device from reported runtimes.

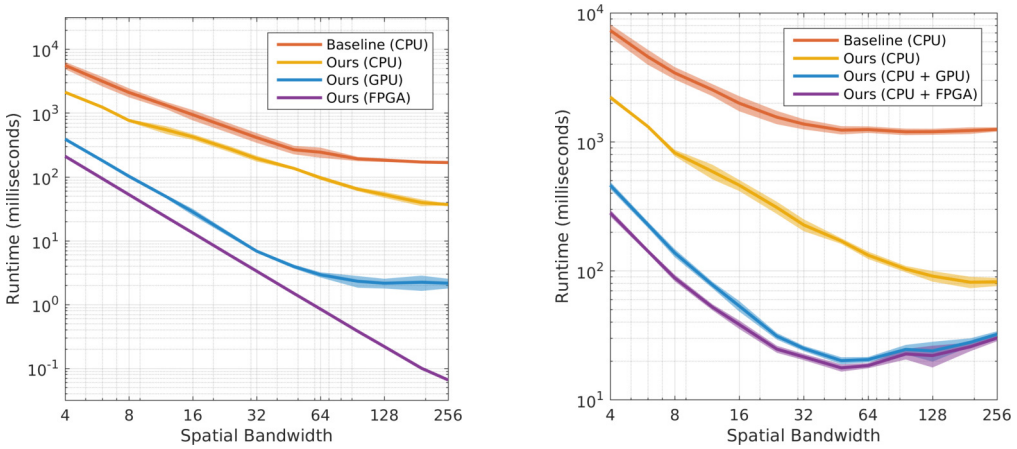
4.4.1 *Experimental Results*

We now evaluate the performance of our algorithm and hardware, including runtime comparison, power consumption measurements, and accuracy evaluation.

RUNTIME RESULTS Figure 4.8 plots the runtime results of bilateral solver implementations on all our benchmarks, as a function of the spatial bandwidth. Figure 4.8a shows the runtime for the optimization portion of the solver, and Figure 4.8b shows the runtime for the complete bilateral solver including pre-processing.

As we increase the spatial bandwidth (σ_{xy} of Eq. 4.8), we see that the overall grid size decreases, and runtimes shorten for all implementations. We find that our algorithm outperforms the baseline on all platforms at all spatial bandwidths. For optimization alone, CPU and FPGA results scale with the grid size, while the GPU results scales until the size of the grid is too small to fully utilize resources. Because splatting and slicing is not accelerated on the FPGA, runtime for the entire bilateral solver does not scale as well at large grid sizes.

Table 4.4 highlights the runtime results specifically for the VR Video use-case, where $\sigma_{xy} = 12$ as in [9], as well as the power consumption of each hardware configuration. Our algorithm's speed outperforms the CPU baseline on all platforms evaluated, and



(a) Runtime (Optimization Only)

(b) Runtime (Total)

Figure 4.8: Runtimes of the baseline, CPU, GPU, and FPGA implementations of HFBS, as a function of the spatial bandwidth (σ_{xy} of Eq. 4.8).

Table 4.4: Runtimes for different variants of the bilateral solver on different hardware for the VR video use-case. Runtimes for optimization by itself and for the entire algorithm (problem construction/splatting, optimization, and slicing) are shown independently.

Algorithm / Platform	Opt. (ms)	Total (ms)	Power (W)
Baseline (CPU)	1322±171	2529 ±271	16
Our Algorithm (CPU)	545 ±77	588 ±77	152
Our Algorithm (CPU + GPU)	49 ±3	78 ±5	245
Our Algorithm (CPU + FPGA)	23 ±1	52 ±3	25

our FPGA accelerator is significantly faster than the baseline while also reducing power consumption.

Note that the CPU-only HFBS runtime reported in Table 4.4 is still far from the real-time requirement of 30 frames-per-second. The GPU and FPGA implementations get very close to real-time for $\sigma_{xy} = 12$, but still do not make it. By selecting $\sigma_{xy} = 32$ and losing some accuracy, both FPGA and GPU implementations meet the real-time requirements.

We also observe that HFBS significantly reduces pre-processing. This is mainly caused by the elimination of the Jacobi preconditioner. The switch to a dense 3D bilateral grid improves available parallelism in the splat-slice routines as well.

COMPUTATION-COMMUNICATION TRADEOFFS We consider the throughput of the data output as the "communication cost" for offloading, and the cost to compute the pipeline block as the "computation cost". We treat the communication cost as fixed for each block; it is simply the cost of offloading the data from each block, as shown in Figure 4.9.

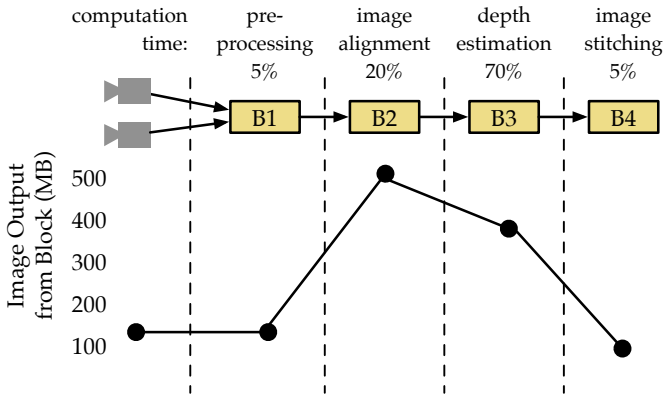


Figure 4.9: Computation distribution and output data size for blocks in a VR video pipeline (2 of 16 cameras).

For all blocks except disparity refinement, we assume the computation cost to be the compute time evaluated using the ARM CPU baseline's performance numbers. We average the compute time for the disparity refinement block over five executions of the kernel over a frame. Because this processing flow can be pipelined across frames in a video stream, the "total cost" of the system can be considered to be dominated by the lowest-throughput block of the system.

Figure 4.10 shows the runtime results of different pipeline configurations, uploaded on a networked connection to a viewing device supporting at least 30 FPS. We seek to uncover scenarios in which both computation and communication surpass our minimum frame rate of 30 FPS—if one or both costs falls below the threshold, the system cannot support real-time operation.

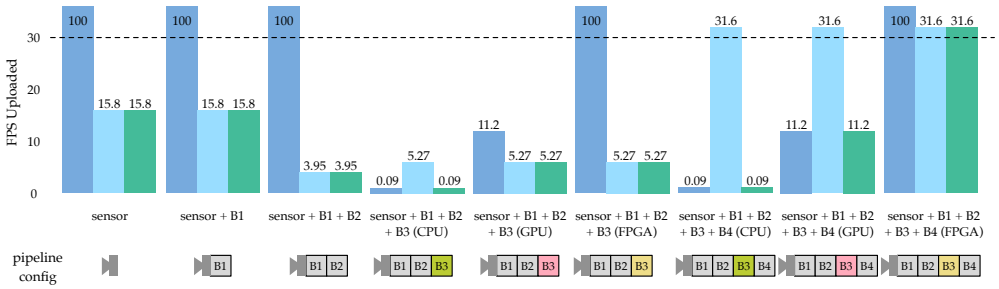


Figure 4.10: Pipeline configurations with different bilateral smoothing implementations (CPU, GPU, FPGA), and resulting upload rates (frames per second). Only the full pipeline with FPGA acceleration can meet a 30 FPS upload requirement.

For the first three scenarios, the cost of doing little computation before offloading is cheap, even on the ARM core, but the communication cost for the raw captured data falls short of our 30 FPS threshold. Computing the disparity refinement in B_3 is more costly, and the CPU and GPU implementations are not fast enough to support real-time

Table 4.5: Depth Superresolution Task [53]

Algorithm	Error (MSE)	Runtime (sec)
Chan et al.[26]	3.89	3.02
Min et al.[127]	3.78	0.383
Domain Transform [62]	3.60	0.021
Ma et al.[119]	3.53	18
Zhang et al.[201]	3.51	1.346
Guided Filter (Matlab) [79]	3.51	0.434
Fast Guided Filter [78]	3.45	0.225
Yang et al.[195]	3.44	0.304
Farbman et al.[50]	3.24	6.11
JBU [3, 99]	3.19	1.98
Barron & Poole [13]	2.75	0.234
Our Model	3.27	0.047 ± 0.002

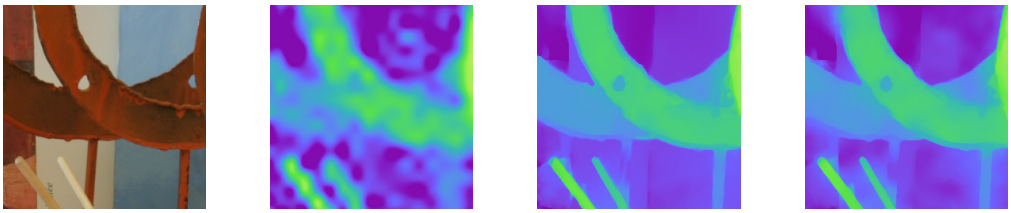
operation. Moreover, the cost of offloading the computed depth maps before image stitching is significantly lower.

The computation cost of image stitching in B_4 is marginal compared to BSSA, as well, and the resulting FPS is virtually the same. The data size to communicate after B_4, however, is much smaller, as discussed in Figure 4.9, and is the only data size small enough to support real-time uploading. We find that the configuration with all the blocks processed in-camera and B_3 mapped to the FPGA is the only configuration where both computation and communication pass the threshold and support real-time processing.

Our analysis indicates that this camera system is primarily constrained by network bandwidth. For our evaluation, we assumed transfer speeds of 25 Gigabit Ethernet. As network connections grow faster, our results will trend towards offloading computation right off the sensor. For instance, at a hypothetical ultra-high-throughput network link of 400-Gb Ethernet, the 16-camera output can be uploaded at 395 FPS, reducing the efficiency incentive for in-camera processing in this scenario.

DEPTH SUPERRESOLUTION Because our proposed model is an approximation to the bilateral solver, we should expect some drop in the quality of our output relative to that of [13]. To quantify this drop in accuracy, we evaluate on the depth superresolution task of [53], which was the primary evaluation used in [13]. We evaluate using the same experimental setup and the same hyperparameters as [13] ($\sigma_{xy} = 8$, $\sigma_l = 4$), and report MSE with respect to ground truth from the Middlebury Stereo Dataset [167].

As can be seen in Table 4.5, our model produces a slightly higher error than that of [13], but has a significantly lower runtime (here we report runtime on a Nvidia 1080 Ti). This increase in error is due to the fact that our model ignores color in the input



(a) Input reference image [167] (b) Input noisy depth (c) Improved depth [13] (d) Our improved depth (HFBS)

Figure 4.11: A qualitative comparison of HFBS's performance compared to the model of [13] on the depth superresolution task of [53]. HFBS produces similar output to [13] and is significantly faster.

image, and so has difficulty distinguishing between pixels with different chroma but similar luma. The images in this task are unusually colorful and "cartoonish", by virtue of being a constructed vision task, so this increase in error represents an upper-bound on the increased error we expect to see in natural scenes. Even with this reduction in error, we see that HFBS is significantly faster than all more-accurate techniques, and significantly more accurate than all faster techniques.

We present qualitative results for this task in Figure 4.11. As discussed in Section 4.2.2, HFBS requires double the iterations to achieve the same accuracy level of OBS, but still performs significantly faster. We can see that our output depths are qualitatively very similar to those of [13], as expected.

4.5 DISCUSSION

There are a number of optimizations our hardware design can integrate for improved performance. Nevertheless, we observe that our design can be practically deployed at both the camera node or in the cloud to enable real-time VR video rendering.

4.5.1 *Accelerator optimizations.*

There are many opportunities to further optimize our hardware design. For instance, our design only accelerates the iterative optimization portion of HFBS. Integrating splat and slice operations into our accelerator, as in [160], would reduce transfer costs from GB-large bilateral grid to smaller MB-sized images and further reduce runtimes. Also, many vertices of the dense bilateral grid begin as zeros and do not need to be processed; intelligently ignoring these zero-valued grid vertices can reduce wasted computation and potentially improve the runtime. Similarly, the wasted memory space from our addressing scheme can be mitigated with increased control logic, which may allow us to maximize the bilateral grid size. We can also more tightly pipeline the three phases of reading, processing, and writing out grid data to reduce latency; these stages currently execute sequentially. To ease prototyping portability, our design used single PCIe channels for reading and writing, which leaves the remaining channels idle. The

board we target, Xilinx Virtex UltraScale+, supports up to 16 PCIe channels that can be leveraged to improve transfer times.

4.5.2 *System specifications for real-time VR video processing platforms.*

While our design can execute bilateral solving under real-time constraints, the bilateral solver is just one step in the Jump VR video rendering pipeline. Moreover, the design we present processes the flow field from a camera pair while the VR video capture system we target processes 16 flow fields from a 16-camera rig. We outline the specifications and cost for a system that could process the full 16-camera input to produce virtual reality video in real-time in Table 4.6.

Table 4.6: Full-system specification for an end-to-end real-time VR pipeline.

Item	Use	#	Unit $	Total $	Max. Power
GoPro	Camera	16	$360	$5,760	N/A
Virtex Ultrascale+	HFBS	16	$2,995	$47,920	≈400 W
Intel i7-7700K	Host CPU	1	$350	$350	≈ 90 W
Nvidia Titan X	Etc. acceleration	1	$1,600	≈250	
Full-System				$55,360	490 W

The monetary cost of deploying such a many-FPGA system in both configurations is high, but the power consumption of our FPGA-based system, with 16 high-end fully-utilized FPGAs, is approximately that of two GPUs. Such power savings can be critical for mobile camera rigs. At the data center level, power constraints are less stringent, but deploying custom hardware for high-bandwidth tasks can still reduce power consumption and operating costs.

4.6 SUMMARY

The hardware-friendly bilateral solver enables scalable, real-time processing of VR video on modern hardware. We explore a hardware-software codesign approach to construct an algorithm that is both faster *and* more accurate than prior work, optimizing algorithm details and hardware implementation together. In particular, HFBS uses a bilateral-space Heavy Ball algorithm and a 3D dense bilateral grid that allows fast and predictable memory accesses. In contrast, the baseline algorithm [13] uses a 5D sparse grid and is about 4× slower than HFBS. We also design an FPGA accelerator for HFBS using reduced-precision fixed-point numbers and customized memory layout. Our CPU, GPU, and FPGA implementations of HFBS are 4×, 32×, and 50× faster than the original bilateral solver. We observe that our FPGA accelerator is more energy-efficient than

comparable CPU and GPU implementations, and can be practically deployed at both the camera node or in the cloud to enable real-time VR video rendering.

5

PERCEPTUALLY-COMPRESSED VIDEO STORAGE AND STREAMING

For decades, video codecs have exploited how humans see the world, for example, by devoting increased dynamic range to spatial features (low frequency) or colors (green) we are more likely to observe. One such perceptual cue, *saliency*, describes where in a video frame a user focuses their perceptual attention. As video resolutions grow, e.g., 360° video and 8K VR displays, the salient regions of a video shrink to smaller proportion of the video frame [171]. Video encoders can leverage saliency by concentrating bits in more perceptually interesting visual areas. Prior work, however, focuses only on achieving bitrate reduction or quality improvement at the cost of complicated, non-portable prototypes designed for a single codec implementation [68, 71, 110, 118]. Exploiting saliency in video compression can ease the burden of video data growth while maintaining perceptual quality. In this work, we address the challenges of storing and integrating this perceptual data into cloud video storage and processing systems.

In this section, we describe Vignette, a cloud video storage system that leverages perceptual information to reduce video sizes and bitrates. Vignette is designed to serve as a backend for large-scale video services, such as content delivery systems or social media applications. Vignette has two components: a compression scheme, *Vignette Compression*, and a storage manager, *Vignette Storage*. Vignette Compression leverages a new saliency-based compression algorithm to achieve up to 95% lower bitrates while minimally reducing quality. Vignette Storage uses a simple API to trigger saliency-based compression when needed, allowing applications to trade off between faster traditional compression and Vignette's smaller video sizes. The system uses low-overhead metadata, can be easily integrated into existing media storage structures, and remains transparent to standard video applications.

Vignette is *not* a new standalone codec or compression standard. Instead, it extends existing, modern codecs to take advantage of the untapped perceptual compression potential of video content, especially high-resolution video served in VR and entertainment settings. As a result, off-the-shelf software and hardware accelerators can decompress Vignette's perceptually compressed videos with no modifications. We implement Vignette as an extension to LightDB [75], a database management system for video. Our prototype of Vignette demonstrates cost savings to cloud video providers and power savings during mobile video playback. Our work thus complements other advances in codec implementations and can easily be implemented with new standards, like AV1.

This chapter makes the following contributions:

- **Systems support for perceptual video compression.** We propose Vignette, a system for producing and managing perceptually compressed video data. Vignette

produces videos that are 80–95% smaller than standard videos, consume 50% less power during playback, and demonstrate minimal perceived quality loss.

- **A forward-compatible perceptual encoding pipeline.** Vignette leverages existing features of modern video codecs to implement perceptual compression, and can be deployed in any video processing system that supports such codecs, such as HEVC or AV1.
- **Custom storage for perceptual data.** Vignette's storage manager efficiently stores and manages perceptually compressed videos and is integrated in a modern video processing database system. Vignette Storage supports both a heuristic-guided search for fast perceptual compression and an exhaustive mode to compute an optimal saliency-based compression configuration.

To our knowledge, this is the first work to consider storage management of perceptually-compressed video information. Using saliency as a motivating perceptual cue, we evaluate the limits of perceptual compression in a video storage system with a collection of modern and high-resolution video datasets. Vignette's compression scheme uses a neural network trained to predict content saliency and an off-the-shelf HEVC encoder to reduce bitrate requirements by 80–95%. Our results show that Vignette can reduce whole-system power dissipation by 50% on a mobile phone during video playback. Quantitative evaluation and user study results validate that these bitrate and power savings come at no perceived loss in video quality.

5.1 BACKGROUND: PERCEPTUAL COMPRESSION USING SALIENCY MAPS

Saliency is a widely-utilized measure of the perceptual importance of visual information. Saliency data encodes the perceptual importance of information in a video, such as foreground and background or primary and secondary objects. Video codecs already use some perceptual information, like motion and luminance, to improve compression performance [174], but new modes of video viewing (such as with a VR headset) introduce the opportunity to integrate richer cues from the human visual system [103]. In this paper, we use saliency as an example of one such perceptual cue to demonstrate the potential of perceptual compression. This section provides background on saliency, compares methods for generating and encoding saliency information, and introduces the machine learning technique Vignette uses to gather perceptual information about video data. We also describe *tiles*, the codec feature we use to compress videos with saliency information.

5.1.1 *Salience Maps and Detection Algorithms*

Saliency-detection algorithms highlight visually significant regions or objects in an image. A saliency map captures visual attention in the form of a heatmap, where the map's values correspond to the salience of pixels in the input. In this paper, we visualize saliency maps as grayscale video frames or heatmaps for clarity.

In the past, saliency information was hard to generate accurately without detailed per-video information, such as hand annotation or detailed eye gaze logs. Moreover, the low latency and poor spatial resolution of eye-tracking devices prevented effective deployment of eye-tracker-based saliency prediction [20]. VR headsets, however, are natural environments for the efficient deployment of eye tracking, and they have motivated improvements in the performance and accuracy of eye trackers [187]. Recent work combining these improved eye tracker-generated fixation maps with deep learning has improved the accuracy of saliency prediction algorithms, especially for natural images and video [22].

5.1.2 *Systems Support for Perceptual Video Compression*

Prior work has investigated many techniques for including saliency information in video compression, reducing bitrate at iso-quality by 20-60%. However, these techniques required significant changes to video codecs, i.e., maintaining full-frame saliency maps to use as additional input [118], computing saliency models on-the-fly at high computational cost [68], or solving complex optimization problems to allocate video bits [110], as the quality of saliency map generation was the limiting factor to deploying perceptual compression. Recent interest in applying machine learning techniques to problems in visual comprehension resulted in accurate saliency prediction models that effectively mimic the human visual system [22]. Moreover, interest from VR developers in deploying fast and accurate eye tracking for improved VR experiences further improved the accuracy of saliency maps with high quality fixation data, leading to a virtuous cycle of saliency map prediction and improvement [187]. The final challenge in closing the gap for deploying perceptual video compression is to design storage systems that manage and support perceptual information.

5.1.3 *Tiled Video Encoding*

Vignette uses tiles to implement perceptual compression. Tiling a video divides a single video stream into independent regions that are encoded as separate decodable streams [128]. Encoders can code tiles at separate qualities or bitrates, and decoders can decode tiles in parallel. Tiles are simple to express using standard encoding libraries, like `FFmpeg` [54] and are supported in many video codecs. Restricting our implementation to native tiling features introduces some loss of detail compared to designing a custom encoder. Standard encoders only support rectangular tiles, and cannot leverage motion across tiles during encoding process. Using only native features, however, guarantees that our compression scheme is compatible with any modern codec that implements tiling, like HEVC or AV1 [10]. As video standards and codec efficiency improve, using general codec features to perform encoding and manage storage ensures that perceptual information remains useful.

5.2 VIGNETTE SYSTEM OVERVIEW

We designed Vignette to be easily deployed in existing video storage systems and transparent to video applications that do not require perceptual information. Figure 5.1 shows how Vignette can be deployed on a typical video storage system, with Vignette Compression used during the transcoding pipeline, and Vignette Storage managing the integration of perceptual information with video data.

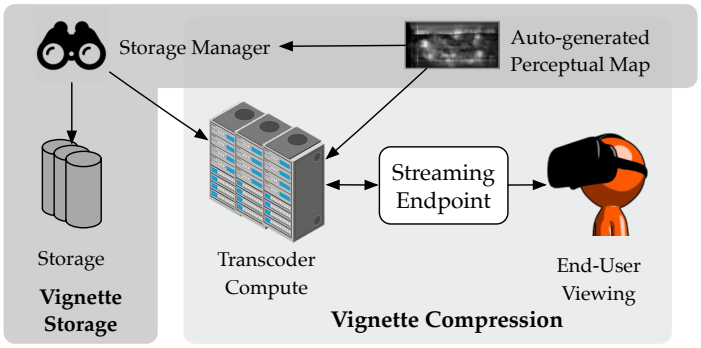


Figure 5.1: Vignette provides two features: Vignette Compression, a perceptual compression algorithm, and Vignette Storage, a storage manager for perceptually compressed videos. Integrating perceptual information with the storage manager reduces network bandwidth and storage costs.

VIGNETTE COMPRESSION uses native features found in modern video codecs. Our implementation of Vignette Compression produces videos that work out-of-the-box with any system that supports HEVC, including hardware accelerators. Vignette Compression perceptually compresses videos by enumerating configurations of video tiles and saliency-quality correspondences to maximize quality while minimizing video size. The algorithm has three high-level steps: generate a perceptual data map (e.g., saliency map) for a given video file (Section 5.3.1), determine the optimal number of rows and columns, which we call a "tile configuration", to spatially partition the video into (Section 5.3.2), and select a mapping of saliency values to encoder qualities, for each tile (Section 5.3.3).

VIGNETTE STORAGE manages perceptual information as simple metadata embedded within videos or maintained in the storage system. This reduces storage complexity for data management and ensures Vignette data is transparent to saliency-unaware video applications such as VLC or Optasia [117]. Vignette Storage can use a neural network to generate saliency information or collect them from end-user video viewing devices. The storage manager supports the following features: low-overhead perceptual metadata transmitted alongside video content, without impeding the functionality of applications that choose not to use it (Section 5.4.2), storage management policies to trigger one-time perceptual compression during "open loop" mode, support for refining perceptual video compression with cues from user viewing devices in a "closed loop"

mode (Section 5.4.3), and a heuristic-based search for faster perceptual compression (Section 5.4.4).

5.3 VIGNETTE PERCEPTUAL COMPRESSION DESIGN

Vignette Compression uses off-the-shelf video codec features to encode perceptual information and improve coding efficiency. Our technique takes a video as input, generates a per-frame perceptual map for the video, and aggregates the per-frame maps into a single video saliency map. Vignette Compression then transcodes the input video with a tiled encoding, where the quality of each tile corresponds to the saliency of the same tile in the video's saliency map. It uses only the native features of the HEVC codec to ensure compatibility with other video libraries.

5.3.1 *Automatically Generating Saliency Maps*

We use MLNet ([39]) to automatically generate a corresponding saliency map for a video input. Figure 5.2 shows the saliency map generated for a video frame and how the generated maps capture the visual importance of a given video frame. MLNet uses Keras with Theano [34, 178] to perform saliency prediction from video frames. The process requires decoding the video and processing each frame through the neural network to produce output saliency maps. We accumulate the per-frame saliency maps into a single map by collecting the maximum saliency for each pixel in the frame across the video file. These aggregated saliency values produce a single saliency map of importance across the video. This method uses much more compute time than a method only generating saliency maps for keyframes or at a set timestep, but more generously accommodates motion and viewpoint changes during a scene. Because video storage systems slice videos into short segments (10-20 seconds) for better coding efficiency, these video saliency maps capture aggregate saliency information without oversaturating the saliency heatmap.

5.3.2 *Leveraging Saliency With Tiled Video Encoding*

Once a saliency map for each video is produced, we then use it to perceptually encode videos with the tiling feature in HEVC [174]. To produce saliency-based tiled video encoding, we divide a video segment spatially into tiles and then map each tile to a quality setting. The saliency map's value at each tile determines the tile's quality setting. For simplicity and generality, the tiling patterns we use are rectangular tiles with uniform width and height across the video frame. We use the same tile configuration throughout the entire 10-20 second video segment for coding simplicity. We select the number of rows and columns in each a tiling pattern based on either an exhaustive search of all tile configurations or a heuristic-guided search, described in Section 5.4.4.

While tiling is simple and provides coding benefits, a given tile configuration can incur overheads from introducing suboptimal encoding boundaries. Tiles are self-contained

video units that can be decoded separately. They cannot compress information beyond per-tile boundaries. As a result, information that may be efficiently coded using partial frames in a standard encoding must be repeated if it appears in multiple tiles. A poor tile configuration produces less efficient videos than a standard encoding pass, especially for fast-moving scenes.

We minimize the penalty of adding tile boundaries in areas that would benefit from being encoded together by exhaustively enumerating all tile configurations. We consider only uniform-sized tiles by evaluating across all row-column pairs a video frame allows. The HEVC standard constrains the minimum size of row and column tiles, which restricts the row-column tile configurations allowed. In practice, we enumerate tile configurations ranging from 2×2 to 5×10 and 10×5, compress the tiles according to their saliency values, and measure the resulting bitrate and video quality achieved. This exhaustive enumeration takes about 30 minutes per 15-second video to find the best tile configuration with our experimental setup.

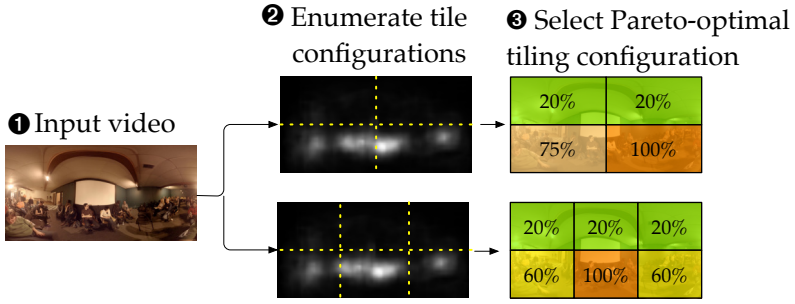


Figure 5.2: Overview of Vignette Compression algorithm.

5.3.3 *Mapping Saliency to Video Quality Rates*

Each HEVC tile is encoded at a single quality or bitrate setting throughout the video stream, requiring Vignette Compression to select per-tile encoding qualities. We deconstruct saliency maps into per-tile parameters by mapping the highest encoding quality to the maximum saliency value in the tile's saliency map. Selecting the video encoding quality that corresponds to a tile's saliency value is less straightforward. Mapping saliency to video quality involves determining how video quality should be expressed during encoding and how saliency should correspond with that quality measure.

HEVC exposes different modes of controlling quality and bitrate, such as constant bitrate or constant rate factor, with varying levels of effort and efficiency. For evaluation simplicity, we use a perceptually-controlled version of a target bitrate, where the target bitrate either corresponds to the bitrate of the original video or is specified by the API call. The highest-saliency tiles in the video are assigned the target bitrate, and tiles with lower saliency are assigned lower bitrates, with a minimum bitrate of 10% the original video bitrate. As shown in Figure 5.2, we encode a 0-255 saliency map as discrete bitrates corresponding linearly from a minimum value to the target bitrate or

quality, which is the maximum. Because Vignette supports standard codec features, target bitrate could be replaced with a codec's quality control, i.e. constant rate factor, as well.

5.4 VIGNETTE STORAGE SYSTEM DESIGN

We now describe Vignette's storage manager for maintaining perceptual video information. Vignette Storage uses low overhead metadata to encode perceptual data and a heuristic-guided search to reduce the compute load of generating perceptual transcodings. Vignette Storage's metadata representation reduces full-resolution frames to a small number of bytes, and its heuristic search algorithm reduces the time taken to find an optimal tile configuration by ~30× in our experiments.

5.4.1 *Overview of Vignette Storage*

Vignette Storage exposes perceptual video compression to applications by providing three features: (1) transparent perceptual metadata, (2) simple storage management policies, and (3) a search algorithm that reduces transcoding cost. We embed perceptual metadata as a side channel within the video container. Standard video containers (i.e., mp4) encapsulate saliency information along with video content, so that applications with and without perceptual support can decode Vignette videos. A 360° video player, for example, can initialize videos to be oriented in the direction of a high-saliency region it decodes from Vignette metadata, but the videos can also be played traditionally in a standard video player like VLC.

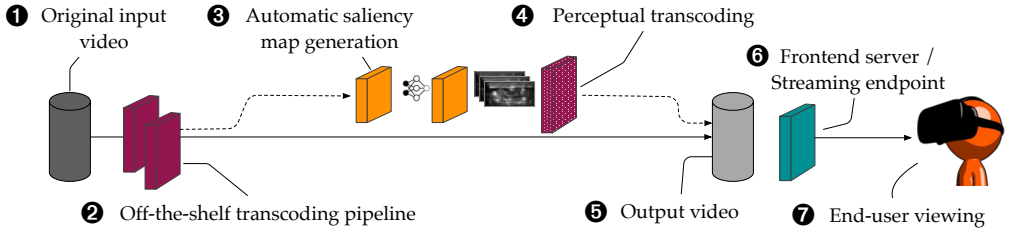


Figure 5.3: High-level architecture of Vignette system design. Vignette automatically generates saliency maps to include perceptual data during video transcoding.

Vignette Storage can be used in both open and closed-feedback loops for perceptual transcoding; Figure 5.3 shows how Vignette Storage can switch between an "open loop" mode, where video is perceptually compressed once based on automatically generated saliency maps, and a "closed loop" mode, where perceptually compressed video can be updated based on cues from user-end viewing devices. The heuristic search feature included in Vignette Storage leverages intrinsic video features to enable ~30× faster perceptual transcoding at near-optimal quality results.

Vignette Storage operates like similar large video management services [85, 116, 121]. Upon upload, it chunks videos into segments, typically 6-12 seconds in length.

Each video segment consists of one keyframe and an ensuing set of predicted frames. Vignette Storage can perform perceptual compression on a per-video basis, or across the video library when a specified condition is met (e.g., low storage capacity, or video popularity decreasing beneath a threshold).

5.4.2 *Saliency Map Metadata*

Video storage systems maintain containers of compressed video data that store relevant video features in metadata. Vignette Storage adopts this approach, and injects a small amount (~100 bytes) of saliency metadata inside each video container. We encode this map as a bitstring that includes fields for the number of rows and columns used for tiled saliency and the saliency weights for each tile. These bitstrings typically range in size from 8–100 bytes. Figure 5.4 shows how this metadata is included as a saliency `trak`, similar to other metadata atoms in a video container.

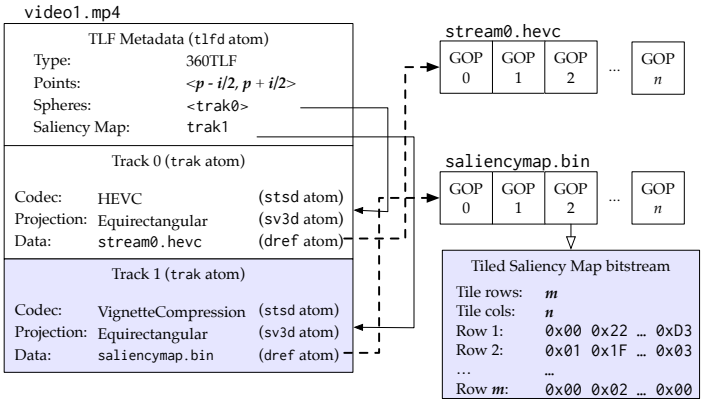


Figure 5.4: Physical layout of video metadata in LightDB. Vignette-specific features are highlighted.

5.4.3 *Vignette Storage API*

The Vignette Storage API defines functions to support the open- and closed-loop modes shown in Figure 5.3. Table 5.1 shows the programming interface for Vignette, which includes three perception-specific operations: `vignette_transcode()`, `vignette_squeeze()`, and `vignette_update()`. Each API operation ingests a video and some required parameters and outputs a video with any generated perceptual metadata encapsulated in the video container.

The Vignette API is included as a shared library linked into LightDB. System developers using Vignette Storage to manage video data can write storage policies or preconditions to execute Vignette Storage functions for a specific video or collection of videos. For instance, a social media service could apply perceptual compression as videos decrease in popularity to reduce storage capacity. A VR video-on-demand

Function	Compression Type	Data required
`transcode`	General	`<IN video, IN CRF/target bitrate, OUT video>`
`vignette_transcode`	Perceptual	`<IN video, (IN CRF/target bitrate,) OUT video, OUT saliency metadata>`
`vignette_squeeze`	Perceptual	`<IN video, IN CRF/target bitrate, OUT video>`
`vignette_update`	Perceptual	`<IN video, IN fixation map, OUT video, OUT saliency metadata>`

Table 5.1: Vignette API

service that ingested eye tracking information could apply perceptual compression as new perceptual information is collected for certain videos.

TRANSCODE FUNCTIONS. Transcode operations express the most basic Vignette Storage function, video transcoding. When a new video is uploaded to the storage system, the storage manager triggers the general-purpose `transcode()` function to transcode the video to any specified bitrates and formats for content delivery. This function takes as input a video and target quality parameter, expressed either by CRF or bitrate, and produces a regularly transcoded video.

The `vignette_transcode()` function is the default saliency-based API call. It takes as input a video and an optional quality or bitrate target, and produces both a video and its corresponding generated saliency metadata. When `vignette_transcode` is triggered, Vignette Storage generates new saliency maps, and then compresses the video according to the target quality expressed.

Vignette Storage's transcode functions use similar signatures, letting the system easily switch between regular and perceptual compression when storage system pressure changes. Including saliency information as a metadata stream included in the video file container makes it transparent to saliency-agnostic applications or commands like `mediainfo` or `ffprobe`.

QUALITY MODULATION FUNCTIONS. As noted in Section 5.3.3, Vignette Compression maps saliency to quality levels for each tile. A `vignette_squeeze()` call will re-compress a video using a specified, reduced bitrate or quality threshold. It takes in a video, target bitrate, and saliency mapping and produces the newly compressed video. For instance, `vignette_squeeze(input.mp4,100k)` transcodes a previously saliency-encoded video from a higher bitrate to a maximum of 100kbps in the most salient regions. The `vignette_squeeze()` function will recompress videos from a higher quality mapping to a lower one, but it will not transcode low-quality videos to a higher-quality mapping to avoid encoding artifacts. This function only executes transcoding and compression with pre-generated saliency metadata, but does not update or generate new saliency metadata. A system can invoke `vignette_squeeze()` before video data is sent to smaller cache or in preparation for distribution to devices with smaller displays.

FUNCTIONS FOR UPDATING PERCEPTUAL MAPS. Vignette Storage also supports a "closed-loop" mode, where saliency maps are updated with new information from eye tracking devices. To invoke this mode, Vignette Storage uses the `vignette_-`

`update()` function to ingest and re-process videos with new perceptual information. A 2-dimensional eye tracker map is easy to convert to the saliency map input used in Vignette Compression. Similar to how Vignette constructs per-video saliency maps, `vignette_update()` updates the video's saliency map with eye tracker information by executing a weighted average of the original map and the input eye tracker map. The update function takes in a fixation map and generates a new metadata bitstream of saliency information that is attached to the video container.

5.4.4 *Heuristic Search for Tile Configurations*

Most of Vignette's computation overhead comes from the exhaustive search over tile configurations for a given video. This exhaustive search is typically performed once, upon video upload, but consumes significant processing time. Vignette Storage contributes a lower cost search algorithm that achieves near-optimal results with a ~30× performance improvement, for situations where fast saliency-based transcoding is required, e.g., for a newly uploaded video. Depending on available resources, a video storage system could choose the exhaustive search for optimal results or heuristic-guided search for faster processing.

Vignette's search technique uses motion vector information from encoded video streams to estimate the size of video tiles. It enumerates tile configurations that group regions of high motion together, and selects a configuration that minimizes the difference in motion vector values across tiles. This heuristic approximates the observation that high-motion areas should not be divided across multiple tiles.

The algorithm extracts motion vector information from encoded videos using the software program MPEGflow [96] and requires one transcoding pass. Similar to our tile configuration search from Section 5.3.2, the search exhaustively evaluates tile configurations of the motion vectors. The search evaluates the motion encapsulated by tiles under a configuration and chooses the configuration with the minimum deviation of motion vectors in each tile. This heuristic approximates the result of exhaustive encoding but uses much less computation. Yet, this technique works well because good tile configurations are able to encapsulate redundant motion or frequency information with a single tile, rather than replicate it across tiles. Compared with an exhaustive search, which can transcode a video hundreds of times to empirically produce the optimal tile configuration, our algorithm produces a result ~30× faster than the exhaustive method and within 1 dB of the best-PSNR result when executed over the videos we use in our evaluation.

5.5 EXPERIMENTAL METHODOLOGY

We implement Vignette by extending LightDB [75], a database management system for VR videos. LightDB lets developers declaratively express queries over large-scale video and uses a rule-based optimizer to maximize performance. Developers can easily

```
Decode("rtp://...")
  >> Partition(Time, 1, Theta, π / rows, Phi, 2π / cols)
  >> Subquery([](auto& partition) {
       return Encode(partition, saliency_mapping(partition) })
  >> Store("output");
```

express HEVC-based saliency encoding in LightDB's query language by combining its `Encode`, `Partition`, and `Subquery` operators:
In this example, `Partition` divides the input video into tiles, `Encode` transcodes each tile with the corresponding `saliency_mapping` value as an argument, and `Subquery` executes the given operation over all the partitioned tiles. We also wrote our object recognition queries for Section 5.6.2 in LightDB to simulate video analytics workloads. To generate saliency maps, we used MLNet [39] with publicly-available weights trained on the SALICON [86], which achieves 94% accuracy on the MIT300 saliency benchmark.

BASELINE: We compare Vignette against the HEVC encoding implementations included with `FFmpeg`. We configure `FFmpeg` with support for NVENCODE [140] GPU-based encoding of HEVC video, as it is supported by large-scale video services and devices [41]. We also implement Vignette Compression on top of `FFmpeg` version `n4.1-dev`, and use the GPU-based NVENC HEVC encoder for tiled encoding. Unless otherwise specified, we target a constrained bitrate using maximum bitrate mode (VBV) to rapidly generate results.

We performed all experiments on a single-node server running Ubuntu 16.04 and containing an Intel i7-6800K processor (3.4 Ghz, 6 cores, 15 MB cache), 32 GB DDR4 RAM at 2133 MHz, a 256 GB SSD drive (ext4 file system), and a Nvidia P5000 GPU with two discrete NVENCODE chipsets.

VIDEO WORKLOAD DATASETS: We use a collection of video datasets, listed in Table 5.2, to evaluate the impact of our techniques across different classes of video. Standard video formats and emerging VR formats comprise our evaluation datasets. The former include representative workloads from Netflix [109] and YouTube [116]. The VR and emerging video datasets highlight demands of ultra high-definition (UHD) formats such as 360° video [115] and the Blender stereoscopic and UHD open source films [60]. To construct a representative sampling of Blender video segments, we partitioned the movies in the Blender dataset ("Elephants Dream", "Big Buck Bunny", "Sintel", and "Tears of Steel") into 12-second segments, and selected five segments that covered the range of entropy rates present in each film.

In this collection of datasets, we found that the `vbench` "desktop" video, a 5-second computer screencast recording, responded poorly during all compression evaluations because of its low entropy and content style, so we excluded it from our evaluation results. We also replaced Netflix's single "Big Buck Bunny" video segment with the same video content from Blender's stereoscopic, 4K, 60 frames-per-second version of the video.

Type	Benchmark	Description	Bitrate (Mbps)	Size (MB)
Standard	vbench [116]	YouTube dataset	0.53–470	757
	Netflix [109]	Netflix dataset	52–267	1123
VR	VR-360 [115]	4K-360 dataset	10–21	1400
	Blender [60]	UHD / 3D movies	10–147	6817

Table 5.2: Video datasets used to characterize Vignette.

QUANTITATIVE QUALITY METRICS: We measured video encoding quality using two quality metrics, peak signal-to-noise ratio (PSNR) and eye-weighted PSNR (EWPSNR). PSNR reports the ratio of maximum to actual error per-pixel, in decibels (dB), by computing the per-pixel mean squared error and comparing it to the maximum per-pixel error. PSNR is popular for video encoding research, but researchers acknowledge that it fails to capture some obvious perceptual artifacts [109]. Acceptable PSNR values fall between 30 and 50 dB, with values above 50 dB considered to be lossless [116]. For saliency prediction evaluations, researchers developed eye-weighted PSNR to more accurately represent human perception [110]. EWPSNR prioritizes errors perceived by the human visual system rather than evaluating PSNR uniformly across a video frame. We computed EWPSNR using the per-video saliency maps described in Section 5.3 as ground truth.

5.6 EVALUATION

We designed our evaluation to answer the following questions:

- **Storage:** What storage and bandwidth savings does Vignette provide? How do tile configurations affect compression gains and quality?
- **Quality of Service:** How does Vignette's compression technique affect quality of service (QoS) of video services like video streaming (perceptual quality user study) or machine learning (speed, accuracy)?
- **Compute Overhead:** What is the computational overhead of Vignette's compression algorithm and storage manager?
- **Data Center & Mobile Cost:** How do Vignette's storage and network bandwidth savings impact video storage system and mobile viewing costs?

5.6.1 *Storage and Bandwidth Savings*

To evaluate the storage and bandwidth benefits of Vignette, we applied Vignette Compression to the corpus of videos described in Section 5.5. We transcoded our video library at iso-bitrate in salient regions and decreased bitrate linearly with saliency to a minimum 10% target bitrate in the lowest saliency tiles, as illustrated in Figure 5.2.

In these experiments, we examine how our transcoding performs across a range of resolutions and workloads, as is expected in a video storage system.

IMPACT OF TILING ON COMPRESSION AND QUALITY: We first examined the impact of tiling on compression benefits using a fixed saliency map. We used an exhaustive tile configuration search and evaluated all tile sizes to identify an optimal number of tiles for each video. We observed that, given a fixed saliency map, optimal tile configurations to maximize storage savings and quality varied based on entropy and video content. Some videos benefited from many small tiles, while others performed best with fewer large tiles.

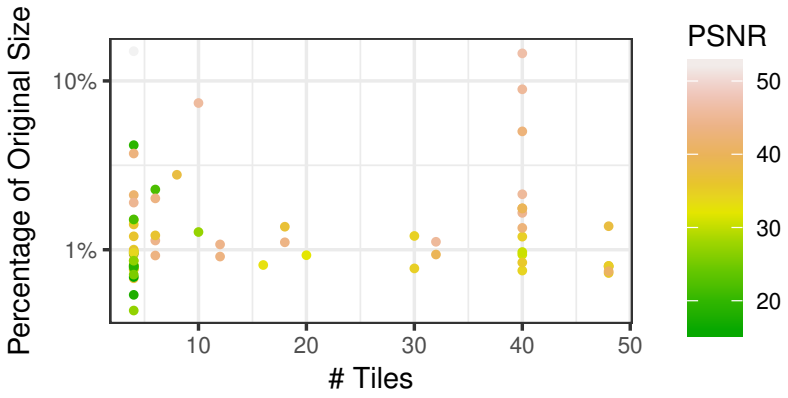


Figure 5.5: Compression ratio and PSNR at the optimal number of tiles for each video. The optimal number of tiles is video-dependent, not correlated with quality or compression ratio.

The smallest tile size we evaluated were 64 pixels in breadth, but most videos performed best with tiles having a breadth of 300–400 pixels. As Figure 5.5 shows, this experiment indicated that the optimal tile configuration for a video is content-dependent and can vary from four tiles to forty, and that tile configuration is an important component of tile-based compression.

OVERALL COMPRESSION, BANDWIDTH, QUALITY: We next explored peak compression, bandwidth, and quality savings by applying Vignette to our video corpus and evaluating compression and quality savings. We used the results of our exhaustive tile search to identify the best compression-quality configurations for each video. Figure 5.6 shows aggregate storage savings, partitioned by dataset.

Overall, we find that Vignette Compression produces videos that are 1–15% of the original size when maintaining the original bitrate in salient regions. These compression savings include the fixed overhead of perceptual metadata, which is <100 B for all videos. Datasets with higher video resolutions (Blender, VR-360) demonstrated the highest compression savings. The vbench dataset, which is algorithmically chosen to have a wide variance in resolution and entropy, exhibits a commensurately large variance in storage reduction. Of the videos with the lowest storage reduction, we find that each

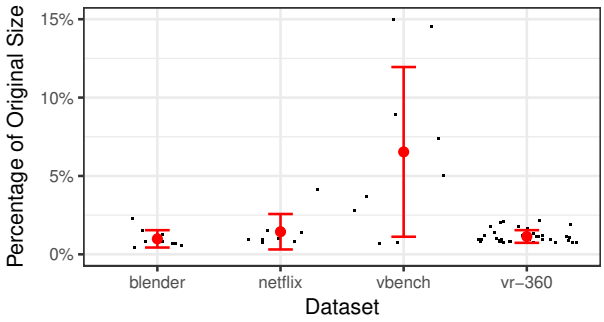


Figure 5.6: Aggregate storage savings by dataset. Vignette Compression reduces videos to 1–15% of their original size while maintaining PSNR of 34–39 dB and EWPSNR of 45-51 dB.

tends to have low entropy, large text, or other 2D graphics that are already efficiently encoded.

Table 5.3 shows the average reduction in bitrate and resulting quality, measured in PSNR and EWPSNR. Our results show that EWPSNR results are near-lossless for each benchmark dataset. The PSNR values—which do not take the human visual processing system into account— are lower, but still above the 35 dB threshold, and so remain acceptable for viewing.

Table 5.3: Average bitrate reduction and quality measurements for Vignette Compression by dataset. For PSNR and EWPSNR, > 30 dB is acceptable for viewing, 50 dB+ is lossless.

Benchmark	Bitrate Reduction	PSNR (dB)	Eye-weighted PNSR (dB)
vbench	85.6 %	39	51
Netflix	98.6	34	45
VR-360	98.8	36	45
Blender	98.2	39	49

Figure 5.7 highlights a Vignette video frame from the Netflix dataset, with an output PSNR of 36 dB and EWPSNR of 48 dB. Overall, the results indicate that Vignette Compression provides acceptable quality for its compression benefit.

5.6.2 *Quality of Service*

To understand the impact of perceptual compression on common video system workloads, we evaluated quality of service (QoS) delivered by Vignette for two applications: entertainment streaming with a user study and evaluation of a video analytics application that performs object recognition. These applications optimize for different QoS metrics: perceptual quality for entertainment video, and throughput and accuracy for object recognition.

(a) Input video frame from Netflix dataset [109].

(b) Saliency map produced by MLNet [39], overlaid on input.

(c) Perceptually-compressed Vignette video, 85% smaller at iso-quality.

Figure 5.7: Example video still, neural network-generated saliency map, and output Vignette perceptually compressed video.

PERCEPTUAL QUALITY USER STUDY: We ran a user study to quantify viewer perception of our saliency-based compression. The study presented users with two versions of the same video: one encoded with HEVC at 20 Mbps, the other with Vignette Compression. The Vignette Compression videos were randomly chosen to be either 1 Mbps, 5 Mbps, 10 Mbps, or 20 Mbps. The study asked users their preference between the matched pairs for 12 videos. The bitrate of the Vignette Compression video varied randomly across the questionnaire. The goal was to discover if viewers prefer Vignette Compression to HEVC, and, if so, if those preferences are more or less pronounced at different bitrate levels for Vignette.

The 12 videos included three videos from each dataset, selected to cover a range of entropy levels. All videos had original bitrates above 20Mbps, except two from vbench. Each video was encoded at a target bitrate (1Mbps, 5Mbps, 10Mbps, or 20Mbps), and the questionnaire randomly selected which bitrate to serve. We distributed the questionnaire as a web survey and ensured videos played correctly in all browsers by losslessly re-encoding to H.264.

We recruited 35 naive participants aged 20–62 (51% women, 49% men) from a college campus to participate in the study. Figure 5.8 shows the results averaged across subjects and videos. When Vignette videos are encoded at 1 Mbps in the most salient regions, 72% users preferred the HEVC baseline. However, for Vignette videos encoded at 5, 10, and 20 Mbps, users either could not tell the difference between HEVC and Vignette, or preferred Vignette videos 60%, 79%, and 81% of the time, respectively. This suggests that video systems can deliver Vignette-encoded videos at 50-75% lower bitrate with little perceived impact.

OBJECT CLASSIFICATION: Video storage and processing systems often perform analytics and machine learning tasks on their video libraries at scale [153, 168, 200]. To evaluate any performance degradation in latency or quality from using Vignette Compression, we profile Vignette while running YOLO [159], a popular fast object recognition algorithm. We compare against baseline HEVC-encoded videos to evaluate if Vignette incurs any additional cost in a video processing setting.

Table 5.4 shows that using Vignette-compressed videos provides some speedup when decoding videos for object recognition, but this benefit is overshadowed by the cost of running YOLO. Examining accuracy, we find that Vignette videos maintain 84%

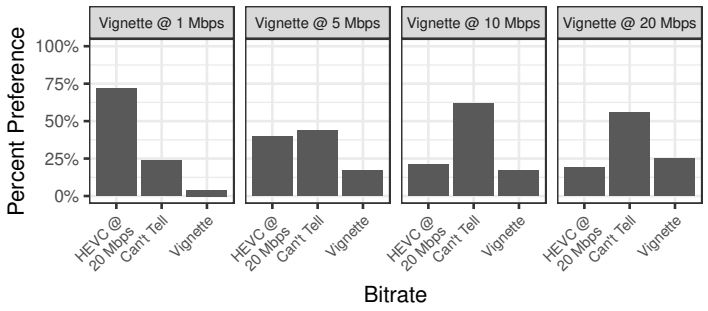


Figure 5.8: Results of perceived quality preference user study, averaged across participants and videos by bitrate. Participants either preferred Vignette or perceived no difference between 20 Mbps HEVC videos and Vignette videos at 5–20 Mbps.

Table 5.4: Vignette Speedup and Accuracy Compared to HEVC Baseline on YOLO Object Recognition.

Decode Speedup	Total Speedup (Decode + YOLO)	Average Accuracy
34.6% ± 14.3%	2.6% ± 2.2%	84% ± 14%

accuracy on average, compared to the baseline HEVC videos. We find that accuracy on the YOLO task is lowest for the videos in the VR360 suite, and tends to correspond to the areas where the video is distorted from the equirectangular projection. While saliency-compressed videos can provide slight benefits for video analytics latency, especially if video decoding is the system bottleneck, future work should investigate how to optimize saliency-based compression for video analytics.

5.6.3 *Compute Overhead*

Vignette Compression bears the additional processing overhead of executing a neural network to generate or update saliency maps. Vignette Storage can switch between an exhaustive or more computationally-efficient heuristic tile configuration search to uncover optimal tile configurations for a video. We benchmarked the latency of the combined saliency and transcoding pipeline in two modes: exhaustive, which generates saliency maps per frame and exhaustively evaluates tiling, and heuristic, which uses the heuristic search algorithm to select a tile configuration within 0.25 dB of the best-PSNR choice (Section 5.4.4).

Table 5.5 shows generating saliency maps in either mode dominates computation time for Vignette, and that our heuristic search is 33× faster than an exhaustive search. This step, however, is only executed once per video and off the critical path for video streaming workloads.

Table 5.5: Mean processing time per video, evaluated over all videos in our datasets.

	Exhaustive		Heuristic	
Task	Time (s)	%	Time (s)	%
Generate saliency map	1633	49%	1633	95%
Compute tile configuration	1696	50	59	4
Saliency-based transcode	21	1	21	1
Total	3350		1713	

5.6.4 *Analytical Model of Vignette Data Center and Mobile Costs*

We use our evaluation results to model Vignette's system costs at scale for data center storage and end-user mobile power consumption. While these results are a first-order analysis, they suggest the potential benefit of deploying Vignette.

DATA CENTER COMPUTE, STORAGE, NETWORK COSTS. Given the high compute cost of Vignette, we evaluate the break-even point for systems that store and deliver video content. We used Amazon Web Services (AWS) prices from July 2018 in the Northern California region to characterize costs. We use a `c5.xlarge` instance's costs for compute, S3 for storage, and vary the number of videos transferred to the Internet as a proxy for video views.

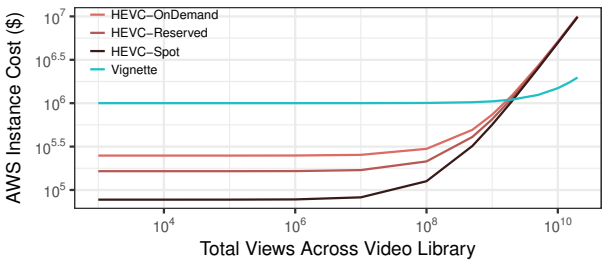


Figure 5.9: Estimated AWS costs for deploying Vignette versus traditional video transcoding. Vignette's additional compute cost is amortized after ~2 billion video views over a 1-million video library.

We assume a video library of 1 million videos that are 10 MB each, encoded at 100 different resolution-bitrate settings (as in [85, 98]) to produce ~500 TB of video data. We estimate baseline compute cost to be a two-pass encoding for each video at $0.212 / sec and Vignette's transcode computation to be 5× a baseline transcode. Larger companies likely use Reserved or Spot Instance offerings, which provide better value for years-long reservation slots or non-immediate jobs; they are 36% and 73% cheaper, respectively. For storage, we estimate costs to be $0.023 / GB on S3 and assume Vignette-compressed videos would be 10% of the original videos (Section 5.6.1). Transferring data out from S3 costs $0.05 / GB; this cost is where Vignette achieves the majority of its savings.

Figure 5.9 shows how different compute pricing models produce different savings at small numbers of video library views, but that Vignette becomes cost-effective at large video viewing rates. For all compute pricing levels, a system would need to service ~2 billion views across a million-video library before Vignette's compute overhead would be amortized across transmission and storage savings. This number is easily reached by large video services; Facebook reported 8 billion daily views in 2016 [126].

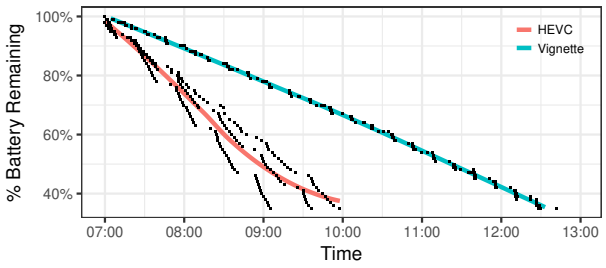


Figure 5.10: Time to dissipate a Google Pixel 2 phone battery from 100% to 30% when viewing HEVC and Vignette videos continuously. Vignette videos provide 1.67× longer video playback on mobile phones.

MOBILE POWER CONSUMPTION. We explicitly designed Vignette to work with the HEVC standard so off-the-shelf software and hardware codecs could decompress Vignette videos. Vignette Compression's tiling strategy, however, makes video bitstream density highly non-uniform across the visual plane. This results in inefficiency for hardware architectures that decode variably-sized tiles in parallel. On the other hand, even such designs will achieve a higher overall power efficiency because of the reduced file sizes to decode and display. To investigate whether Vignette videos can achieve power savings, we profiled power consumption on a Google Pixel 2 phone during video playback of Vignette videos and standard HEVC-encoded videos.

We measured battery capacity on a Google Pixel 2 running Android 8.1.0, kernel 4.4.88-g3acf2d53921d, playing videos on MX Player v1.9.24 with ARMv7 NEON instructions enabled. When possible, MX Player used hardware acceleration to decode videos.[1] We disabled nonessential display and button backlights, as well as any configurable sensors or location monitors, to minimize extraneous power consumption. We logged battery statistics each minute using 3C Battery Monitor Widget v3.21.8. We conducted three trials, playing the 93-file video library in a loop until battery charge dissipated from 100% to 30%, for our HEVC baseline and Vignette videos.

Figure 5.10 shows our results. We found that Vignette video enabled 1.6× longer video playback time with the same power consumption, or, ~50% better battery life while viewing a fixed number of videos. While hardware decoder implementations are typically proprietary, these results indicate that perceptual compression has benefits for mobile viewers, as well as cloud video infrastructure.

1 MX Player only supported decoding stereoscopic videos with the software decoder.

5.6.5 *Discussion*

USING VIGNETTE IN OTHER STORAGE SYSTEMS. We crafted the policies, metadata extensions, and encoder design to make the ideas behind Vignette Storage compatible with pre-existing video storage or transcoding systems. Vignette can easily be implemented for other codecs, like the upcoming AV1, For instance, an Amazon MediaConvert instance with a storage layer in Amazon EBS and Glacier can easily use the policies and metadata in Section 5.4 to implement Vignette. We could further improve Vignette by building on other optimizations that work with off-the-shelf video standards. For instance, Vignette's heuristic search algorithm could include power and performance information from open-source video transcoding ASICs [120, 198] to target more power-efficient tiling configurations. VideoCoreCluster [114] demonstrated energy-efficient adaptive bitrate streaming in real-time using a cluster of low-cost transcoding ASICs, which Vignette could leverage for better server transcoding performance.

OTHER VIDEO SYSTEM OPTIMIZATIONS. Using Fouladi et al.'s parallel cloud transcoding could also improve Vignette's transcode latency, and Vignette's saliency-based tiling could integrate with their codesigned network transport protocol and video codec to better tune streaming quality [58, 59]. At the physical storage layer, Jevdjic et al.'s approximate video storage framework, which maps video streams to different layers of error correction, could be coupled with Vignette's saliency mapping for more aggressive approximation of non-salient video regions [90]. Integrating Vignette with these systems could further improve power efficiency during playback, transcoding latency, or archival video storage durability.

5.7 SUMMARY

This section proposes integrating perceptual compression techniques with cloud video storage infrastructure to improve storage capacity and video bitrates while maintaining perceptual quality. Vignette combines automatic generation of perceptual information with a video transcoding pipeline to enable large-scale perceptual compression with minimal data overhead. Our storage system supports a feedback loop of perceptual compression, including updates as an application gathers data from sources such as eye trackers. Our offline compression techniques deliver storage savings of up to 95%, and user trials confirm no perceptual quality loss for Vignette videos 50-75% smaller in size.

Our limit study of video transcoding shows that significant opportunity for compression can be uncovered by the inclusion of perceptual information. Vignette describes a feasible implementation for managing perceptual information and leveraging it to reduce video data size as more perceptual data is collected. Vignette's design complements the contributions of existing large-scale cloud video storage and processing systems. Video systems can use Vignette to further improve storage capacity or in anticipation of video workloads that produce perceptual information. As users adopt new

technologies to capture more perceptual cues, Vignette's techniques can be extended to utilize these perceptual cues as well. As VR video consumption and new perceptual markers—such as eye trackers in VR headsets—grow in popularity, Vignette's techniques will be critical in integrating perceptual compression at large scale for higher quality, lower bitrate video.

6

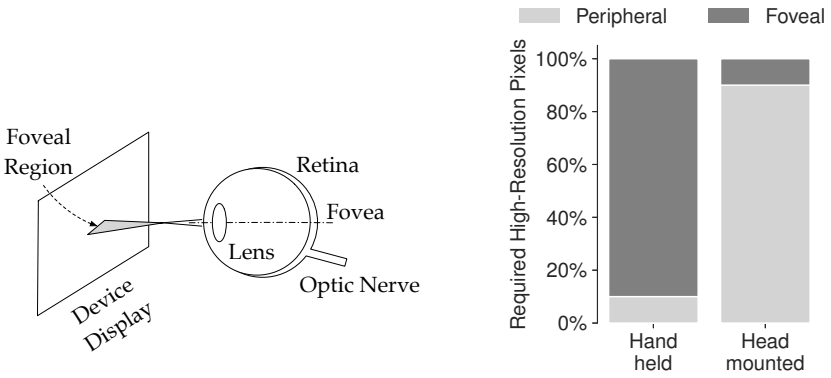
PERCEPTUALLY-COMPRESSED MOBILE VIDEO HARDWARE

Video decoding is a critical motivating workload for virtual reality (VR) headsets. The most popular VR applications, from 360° video tourism and sports to immersive video journalism from the New York Times and National Geographic, all require immersive VR video playback [52, 136]. Recent trends in social VR applications motivate not only watching 3D-360° VR content through these high-resolution headsets, but also streaming traditional 2D video content in VR (e.g., Netflix Party, social YouTube video streaming) [149]. Rendering video for VR displays requires delivering high-resolution (up to 16× more pixels than traditional video) and high frame rate rendering (4-8× 2D video) while running energy-efficient hardware designed to accommodate head-worn video playback without heavy physical weight or heat dissipation. To address these competing constraints, mobile architectures use tightly optimized system-on-a-chip (SoC) platforms with hardware video decode accelerators for video playback. These accelerators support modern video compression standards like HEVC or VP9. However, computation for decoding compressed videos consumes over half of the energy needed for video playback [18, 180]. The growing resolution and size of VR video further increases this energy cost.

VR device constraints and VR video bandwidth demands suggest that video decoder hardware is ripe for VR-specific optimizations. To improve energy efficiency, VR software developers have reduced video data bandwidth by up to 75% by developing *perception-aware* video compression algorithms [67, 76, 77, 150, 161]. These optimizations produce VR video experiences with perceived 8K-quality on 4K displays without commensurate increases in power and bandwidth, and they are currently deployed in VR headsets and software [83, 97, 150]. Architects have proposed adding energy-efficient hardware for other aspects of the VR video pipeline to operate alongside the video decoder [105, 106, 191, 197]. While this wave of innovation in software and hardware improve the stack around video decode hardware, video decoder architectures remain fixed.

Prior work shows that the compute portion of the video decode accelerator alone uses 42-50% of the energy needed, even with optimal data compression [18, 179, 202]. Despite ongoing improvements to software compression algorithms, the decode computation still consumes a significant fraction of the total energy needed for VR video playback. Moreover, VR hardware designs do not directly optimize for the heterogeneous characteristics of VR-specific video formats. As social and AI-augmented video applications become more popular in VR, reducing this fixed cost of decoding compressed video enables more complex video applications without large energy consumption.

This chapter revisits the performance of video decoders in the context of energy-efficient VR video, where evolving software workloads and new hardware blocks motivate flexible, application-aware video decoding. We rearchitect the *video decoder*



(a) The human eye only perceives the foveal region at full resolution [185].

(b) Foveal region of display devices [150].

Figure 6.1: When viewing a VR display at close distance, only a small fraction of pixels (< 10%) are foveal and viewed at high resolution. VR video compression exploits this feature with *foveated compression*.

accelerator to close the performance gap for perceptually-compressed VR workloads with the design of mVDO, a new video hardware decoder. To reduce energy consumption, mVDO introduces *heterogeneous parallel cores* for video decode accelerators and features a new hardware scheduler and decoder configurations optimized for VR video workloads. We codesigned the video decoder architecture in concert with VR video data patterns, mapping peripheral video regions to decoder cores in multiple frequency domains to achieve better energy efficiency. To evaluate mVDO's performance, we used two methods for compressing video perceptually: tile-based foveation and multi-layer foveation. mVDO demonstrates that even small changes in the scheduling of video decoding work can improve energy efficiency for VR video workloads.

mVDO's key optimizations stem from *leveraging irregularity* in perceptually compressed VR video. Perceptual compression techniques exploit constraints in the human visual system to reduce memory bandwidth and associated rendering costs. The central insight is that the human eye perceives only a small portion of the VR display, *the foveal region*, in high resolution, but it still requires peripheral pixels to *approximate* the original image (Figure 6.1). Though software developers have applied this insight to create optimizations that reduce data bandwidth, fixed-function video decoder architectures have not adopted these advances, suggesting an opportunity for up to 50-75% improvement in decode cost commensurate with the data reduction.

In this chapter, we propose mVDO, a video decoder architecture for supporting energy efficient decoding of perceptually-compressed video. mVDO's optimizations target reduced power consumption on VR-centric mobile SoCs for video playback. We reverse-engineered video decode performance for a commodity video decoder and propose new optimizations for perceptual video. We capture our characterizations in mVDOprof, an open-source profiling framework for mobile video decoders. mVDOprof abstracts away codec-specific complexity so compression engineers and VR application developers can more easily assess mobile energy consumption for video workloads.

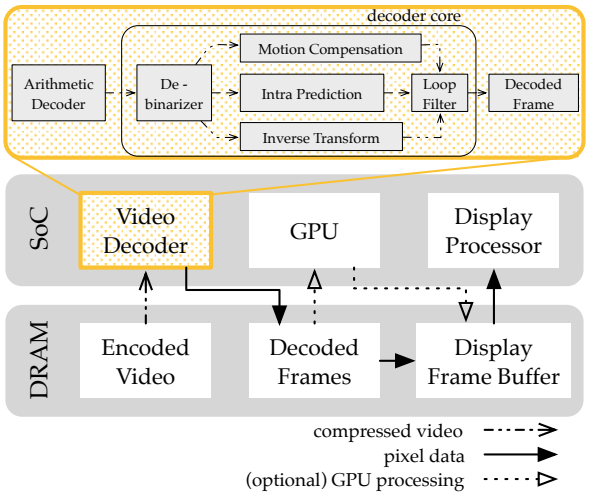


Figure 6.2: System architecture of typical video playback flow on mobile/VR SoC.

This chapter proposes optimizing video decoder architectures for VR workloads. It makes the following contributions:

- mVDO, a new video decoder architecture optimized for tile-based parallelism in VR video, using a new scheduler and heterogeneous decoder cores to improve energy efficiency
- mVDOprof, an open-source profiling framework to characterize video energy consumption on real and modeled video decode hardware to help developers of new perceptual compression techniques leverage architecture-specific energy information to tune their designs

Our evaluation demonstrates that mVDO reduces energy consumption by up to 4× compared to traditional hardware decoders on perceptually compressed VR video and reduces the average energy expended per frame by 12-22% on parallel, frequency-scaled cores.

6.1 BACKGROUND: VIDEO COMPRESSION AND HARDWARE DECODERS

This chapter focuses on playing video in VR distributed using traditional 2D video compression methods, e.g., HEVC or VP9, in accordance with VR video distributors like YouTube and Oculus. In this section, we explain in detail VR video perceptual compression using a traditional codec (HEVC [174]). We focus on HEVC because of its relationship with the baseline hardware architecture used, but our analysis and insights also apply to other codecs, like VP9 [133], H.264 [188], and AV1 [10]. Although specific performance and implementation details vary across codecs, each uses similar data structures and decoding processes. We first discuss the compressed video data structures used (Section 6.1.1). We then give an overview of a generic video decode accelerator architecture at a high level (Section 6.1.2). We finally provide some design details of *video tiles*, a codec feature used heavily for perceptual compression (Section 6.1.3).

6.1.1 *HEVC Data Structure Overview*

Compressed HEVC videos consist of individual blocks, called coding tree units (CTUs) [174]. These CTUs constitute the core representational unit in HEVC. They can vary in size from 16×16 pixels to 64×64 pixels and internally contain units of different types and sizes, such as motion predictions. More detail on the HEVC data structure can be found in the codec specification [174].

Some CTUs reference earlier CTUs in the encoded bitstream for a frame to represent motion or a repeating block. For instance, some motion predictions can reference earlier instantiations in other CTUs. This dependency can occur only in raster order; i.e., CTUs can only reference other CTUs that are spatially above and to the left of them. To accommodate this dependency, CTUs are decoded in row-order. In video decoder architectures with multiple cores, CTUs can be split into parallel rows, or slices, to be decoded simultaneously. Parallel decoding incurs an added delay to account for the spatial dependencies [180]; we analyze further in Section 6.3.

6.1.2 *Video decode accelerator microarchitecture*

Video decoder chips ingest a compressed video bitstream and decode the bitstream into discrete video frames for display or GPU processing. Video decoders are typically deployed on mobile devices as a fixed-function ASIC. Figure 6.2 presents a high-level video decode accelerator architecture for HEVC.

The video decoder initially reads compressed video data from memory. An arithmetic entropy decoder then decodes the compressed data. It produces syntax elements, or individual data elements (e.g., motion predictions, other coding units), to be processed by the appropriate processing unit. Processing units can be grouped together and considered to be a single "decoder core" that operates at a per-CTU granularity, as shown in Figure 6.2. Some hardware decoders instantiate multiple decoder cores to execute in parallel [179] or run a single decoder core at a high clock frequency [202] to decode large videos at fast speeds. CTU decoding is typically pipelined across the stages shown in Figure 6.2's decoder core [202], but a decoder core must wait for any CTU dependencies to finish before decoding. After an entire frame is decoded, the video decoder chip writes it back to the frame buffer, where it can be read by the GPU for further processing or written to the display frame buffer for rendering.

6.1.3 *Video Tile Design Details*

Recent codecs introduced the notion of *tiling* video content [129]. With the growth in popularity of virtual reality headsets and 360° streaming, tiled video has reduced streaming bandwidth and data movement on mobile devices. Perceptual compression schemes partition videos into tiles and use tile-specific parameters to encode quality and bitrate on a per-tile basis [76]. Unlike traditional HEVC tiles, perceptually compressed tiles are not regular streams of tiles with evenly distributed bitrate; instead, they leverage

application-specific information about how VR video is consumed (e.g., where the viewer is looking) to produce *irregular*-bandwidth tiling patterns. Software developers intend for these videos to be decoded by hardware decoders on the mobile SoC, but today's hardware decoders do not optimize for irregular-bandwidth tiling workloads.

The HEVC standard specifies that splitting an encoded video into tiles effectively constructs multiple neighboring video streams, encoded separately, from a single compressed video. Each tile has its own contiguous set of CTUs. Though CTUs in a video stream can reference earlier CTUs, tiling constraints break this dependency: CTUs within one tile cannot reference CTUs in any other tile. This constraint imposes overhead in video coding by making tiles duplicate motion vectors or blocks that appear across tiles, reducing compression opportunities.

Despite overheads, tiling features also introduce new opportunities for parallelism. By partitioning a single video stream into multiple tiles, both video encoding and decoding can be executed with parallel cores. Software decoders can decode tiled video streams in parallel, by using, multiple threads or cores, but hardware decoders cannot decode in parallel unless they have multiple cores to enable parallelism. To facilitate parallel hardware decoding, the HEVC standard includes a bitstream with the locations and entry points for each tile in the compressed video. We leverage this bitstream in the design of mVDO.

6.2 MOTIVATION: IRREGULARITY IN PERCEPTUALLY-COMPRESSED VIDEO

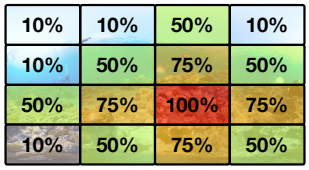
This section describes two popular perceptual compression schemes and characterizes their energy consumption and scalability gap. The two compression schemes are tile-based foveation and multi-resolution foveation. We describe each technique and evaluate performance on a mobile SoC from Nvidia using its hardware video decoder; Section 6.4 lists a complete experimental setup.

6.2.1 *Foveated video compression*

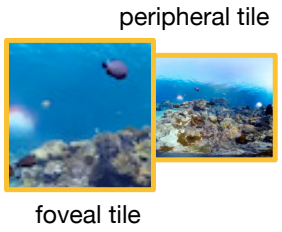
Perceptual video compression reduces memory bandwidth and associated rendering costs by incorporating information about the human visual system. The most faithful method for applying perceptual compression is *foveated compression*. Foveated compression mimics the human eye's performance in the center of the eye, or *foveal* region, with resolution declining exponentially for pixels not at the center of the eye (Figure 6.1a). The point where the eye gaze is centered is the *fixation point*, and the region of pixels around it that maintains the highest resolution is called the *foveal region*, corresponding to the area around the center of the eye's retina, the fovea.

True foveated videos are significantly smaller than conventional videos. They use sparse samples and upsampling to best emulate a true foveated viewing experience [150]. However, they require custom hardware or high-energy GPU processing to decompress and still present challenges related to temporal aliasing and visual artifacts [97]. As an alternative, video distribution services have designed different ways to approximate

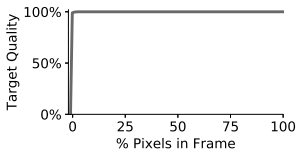
(a) Input 4K frame from VR-360 dataset [38]. Original size is 19 MB.



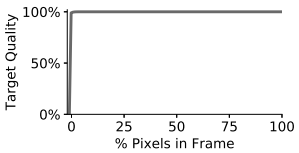
(b) Tiled compression, centered around the fovea. Video size is 11 MB.



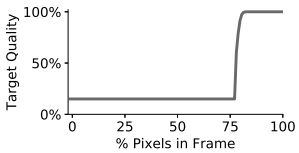
(c) Tiled compression, centered around the fovea. Video size is 11 MB.



(d) CDF of quality target for pixels in frame.



(e) CDF of quality target for pixels in frame.



(f) CDF of quality target for pixels in frame.

Figure 6.3: Example video still and visualizations of tiled and multi-resolution video compression. These VR video compression methods use 40-75% less storage by distributing quality across video tiles.

a true foveated user experience. These perceptual compression methods discard or downscale visually unimportant areas of the image, e.g., peripheral areas of the video frame outside the foveal region [15, 76, 161]. In this chapter, we use the term *perceptual compression* to describe any technique that mimics a foveated viewing experience.

TILE-BASED FOVEATED COMPRESSION. The most popular method of foveated video compression is *tile-based* compression, where many small tiles encode regions of different quality (Figure 6.3). This method approximates the foveated resolution degradation by partitioning the video into rectangular regions with resolution correlated with the distance from the foveal region. Tile-based compression divides the video into contiguous tiles encoded at qualities matching the foveated pattern. In this scheme, the bitrate B for each tile is determined by

$$B \leq c_r H W b_{pp} f_r \tag{6.1}$$

where H and W are the height and width of the video, b_{pp} the bits per pixel–set to 12 as in [97], c_r the compression rate, and f_r the frame rate. For a set number of quality levels l, the bitrate is distributed as

$$B = w_1 B_1 + w_2 B_2 + ... + w_l B_l \tag{6.2}$$

where $w_1 ... w_l$ are weights corresponding to the number of tiles allocating that bitrate. For instance, the toy example in Figure 6.3b has $l = 4$ foveation quality levels, and $c_r = 1.0, .75, .5, .1$.

Foveated tiling (Figure 6.3b) produces larger videos than true foveation, but it can be decoded using traditional video decoder hardware. Tile-based compression can be considered a coarse mode of foveated compression; instead of dropping single pixels in a continuous function, it downscales contiguous tiles and compresses at different scales. Recent work tiles videos into as few as 4 and as many as 128 tiles to improve network bandwidth and streaming latency.

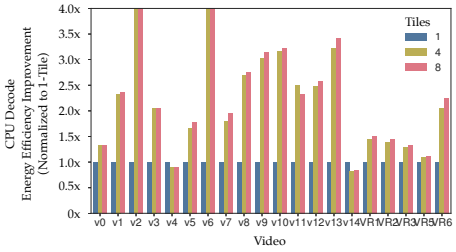
MULTI-RESOLUTION COMPRESSION. This second technique more aggressively compresses video by partitioning the frame into two layers: one small, high-quality tile for the foveal region that is layered over a larger background tile for the peripheral region [67] (Figure 6.3c). Based on visual perception parameters from [67], multi-resolution allocates two bitrate regions $B = w_1B_1 + w_2B_2$, where $w_1 \ll w_2$ and $B_1 \gg B_2$.

The multi-resolution technique has significantly better compression efficiency than the tile-based method but less granularity in controlling the amount of foveation available. On an implementation-level, it can be understood as a minimum-viable foveation method: it leverages human visual models to dedicate the minimum number of bits required for necessary resolution at the cost of the more subtle gradation provided by tiling methods. Recent work deployed multi-resolution foveated compression using two tiles, one for the low-resolution region and the second for the high-resolution, foveal region [15]. However, the size and bitrate of these two tiles are extremely uneven.

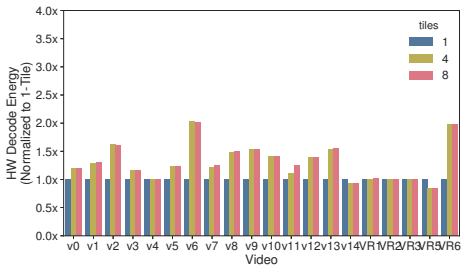
6.2.2 *Memory-compute tradeoffs for perceptual compression*

Despite the recent deployment of tiled and multi-resolution compression, hardware video decode performance of these workloads is not well understood. To motivate the design of mVDO, we first characterized perceptual video decode performance on the hardware video decoder of the Nvidia Jetson TX2 board, the basis for recent VR headsets, e.g., the VR device from Gameface Labs [101]. The TX2 board contains a state-of-the-art Tegra X2 mobile SoC [141], and its design is similar to the smartphone-grade SoCs used by other VR device vendors (like Samsung and Facebook Oculus). We used the TX2's on-board Texas Instruments INA 3221 voltage monitor IC to conduct detailed power measurements of the video decode chip. We disabled WIFI and display rendering to isolate energy consumption related specifically to the video decode accelerator. We ran the TX2 in MAX-Q mode [142] to maximize power efficiency. We activated the hardware decoder through `FFmpeg` [54] while decoding videos from two video benchmark datasets (described in 6.2). We report normalized values averaged across the datasets.

COMPARING FOVEATED VIDEO DECODE ENERGY CONSUMPTION AND FILE SIZE: We first compared the video decode energy consumption and file size of perceptually compressed videos with their traditionally compressed baselines. We tested a 16-tile tile-based foveation pattern, with one tile at 100% bitrate, four tiles at 75% bitrate, and the rest at 50% bitrate. We then tested decoding the videos with the hardware decode accelerator



(a) Energy consumption decoding 1-, 4-, and 8-tile videos on a 4-core CPU, normalized to 1-tile energy.



(b) Energy consumption decoding 1-, 4-, and 8-tile videos on a hardware video decoder, normalized to 1-tile energy.

Figure 6.4: When decoding videos with multiple tiles, hardware accelerators are not as capable as CPUs at capturing this tile parallelism to improve performance.

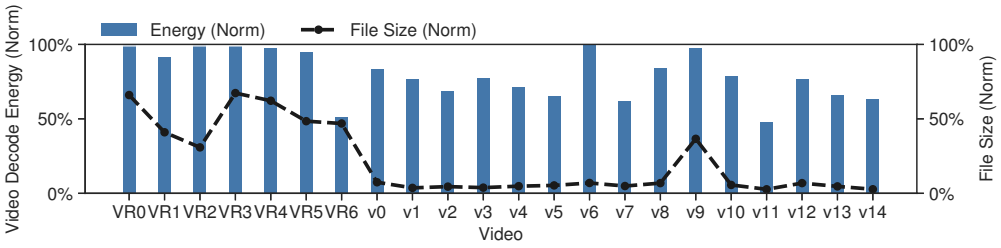


Figure 6.5: Perceptually compressed video file size and decode energy consumption on a Nvidia Jetson TX2 hardware decoder. Perceptually compressed video files are 5-65% of standard videos, but the energy consumed on a hardware video decoder, however, does not scale commensurately with file size reduction.

on the Jetson TX2, measuring energy consumption. We report energy consumption averaged over three trials. 6.5 shows the resulting energy consumption and file size for each perceptually compressed video, normalized to the energy consumption and file size of the original. Though the video file size reduced to 5-65% of the original size with perceptual compression methods, the decode energy consumption rarely scales commensurately. This demonstrates that video decode chips do not exhibit energy consumption changes from reducing compressed video size.

COMPARING FOVEATED VIDEO DECODE ENERGY CONSUMPTION ON CPUS AND ACCELERATORS To further understand video decoder performance on tiles, we examined how the video decoder hardware scaled performance when decoding videos with increasing numbers of tiles. We produced versions of our benchmarks with 4 and 8-tiles, distributing the bitrate equally across tiles to isolate how the decoder core executes on tiles without considering the irregularity of foveated tiles. We measured energy consumption of the CPU decoding the videos using `FFmpeg`'s software HEVC decoder on the 1-, 4- and 8-tile videos. We compared this performance scalability to performance of the hardware decode accelerator on the same workload. For this characterization, we compared the Jetson's on-board hardware decoder with its on-board Quad-Core

ARM Cortex-A57 MPCore. To ensure a fair comparison, we include the controlling CPU energy consumption in the hardware decoder experiment's total energy consumption count. Figure 6.4 shows how each hardware substrate performs on 1-, 4-, and 8-tile videos.

By partitioning the videos from 1 to 4 or 8 tiles, we introduced a new opportunity for parallelism in the decode process. Figure 6.4a shows that when increasing the number of discrete tiles from 1 to 4 tiles, the CPU is able to improve energy efficiency 1.5-4× in most cases. When increasing from 4 to 8 tiles, however, the CPU does not further scale performance. Given that the CPU only has four cores, this reduced scaling makes sense.

Figure 6.4a shows that this performance scalability does not manifest for this hardware decode accelerator. Decode energy consumption only increased by over 50% in 23% of videos tested. This demonstrates decode inefficiency: the decoder improved slightly from the introduction of tiled parallelism, but not nearly commensurate with the improvement a multi-core CPU demonstrated.

ANALYSIS: We find video decoders do not optimize decoding for VR-centric forms of video compression. Further, current video decoder architectures do not effectively scale energy with increased numbers of tiles. As a result, while VR video compression methods reduce memory traffic, decode performance cannot leverage these improvements. These findings motivate an architecture that (1) can scale video decode for multiple tiles and (2) efficiently manage energy consumption and runtime latency for smaller video tiles.

6.3 MVDO: HARDWARE SUPPORT FOR PERCEPTUAL VIDEO DECODING

This section describes mVDO, a new video decoder that optimizes for tile-level parallelism in VR video workloads and tile-proportional execution. It enables perceptual decoding by: adding custom logic to support scalable tile decoding and heterogeneous decoder cores to handle the non-uniformity of perceptually compressed videos.

Figure 6.6 shows a prototype design for mVDO, which builds on a recent low-power video decoder architecture for VR devices [179]. To rapidly explore the design space, we constructed a simulator based on this baseline architecture and added support for our custom scheduling and dual-frequency zones for the decode cores. We optimized decode latency for tiling and multi-layer video and assessed potential speedup from optimal scheduling (Section 6.3.1). We then explored the optimal number of parallel cores needed to efficiently decode tiled and multi-resolution perceptual video (Section 6.3.2). Having selected an appropriate number of parallel cores for a mobile VR video decoder, we considered the energy and latency impact of partitioning the decode units into separate frequency domains to balance tile irregularity across our design (Section 6.3.3).

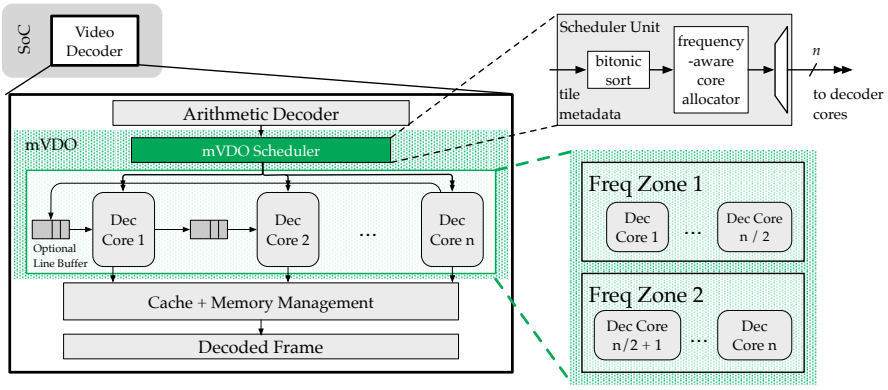


Figure 6.6: mVDO accelerator with heterogeneous parallel decoder cores. mVDO uses a custom scheduler optimized for processing tiled VR video to efficiently distribute work to decode cores. mVDO maps video tiles to decode cores in different frequency regions to further improve energy consumption, allocating tiles based on the complexity of the video tile.

6.3.1 *Scheduling tile decode for lower latency*

mVDO optimizes hardware decoding for tile-based compressed VR video. To understand the change in performance when decoding tiles as opposed than sequential rows, we first present an example using four decoder cores, as in the original baseline decoder architecture [180].

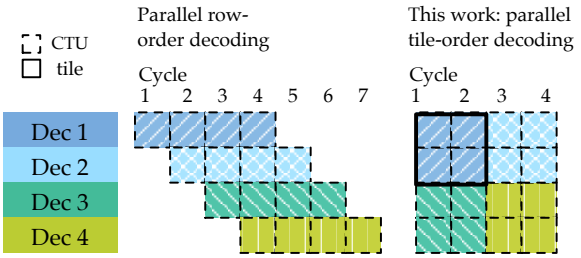


Figure 6.7: Decoding *tiles* in parallel, rather than *rows*, removes row-dependencies.

For a conventional hardware decoder architecture, parallel decoder cores must still abide by dependencies on previous rows and columns, as shown in Figure 6.7. Because tiles break this dependency, decoders can execute on multiple regions without synchronization. This allows decoding to execute with reduced latency. The short example in Figure 6.7, for instance, terminates in 7 cycles for the sequential decoding process, but it can be reduced to four cycles with tile-based decoding. For large-resolution video where complex 64×64 CTUs can bottleneck decoding, these gains can be more significant.

However, the irregularity of data within a tile introduces more inefficiencies. Figure 6.8 illustrates how a naive parallel schedule for distributing tiles across multiple cores can result in unequal distribution of work across the cores. In this example, the default schedule distributes tiles in raster order because they are decoded from the HEVC

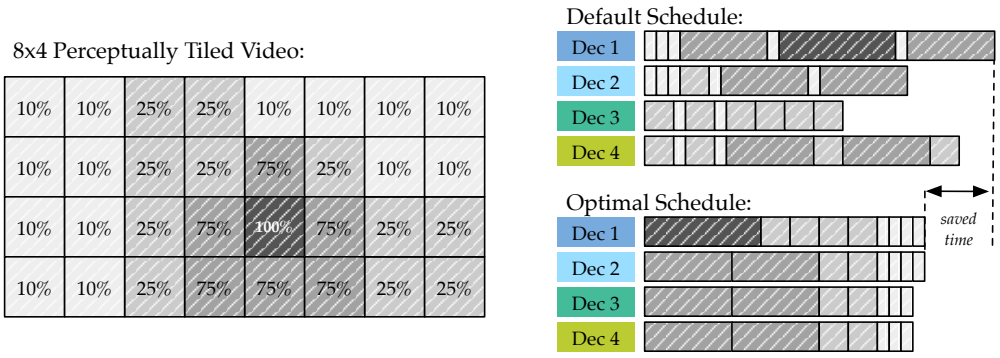


Figure 6.8: Optimal scheduling for tiled video with heterogeneous bitrates. Knowledge of per-tile decode complexity can improve performance.

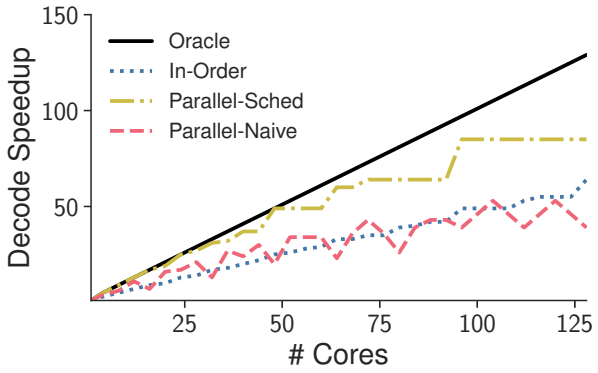
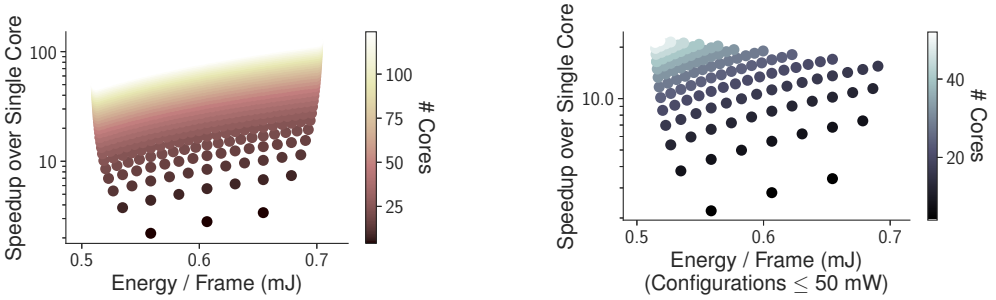


Figure 6.9: Decode performance for in-order and mVDO's parallel decoding, with and without optimal scheduling.

bitstream. This ordering considers only the position of tiles in a frame, even though tiles have no dependencies and can be decoded in any order. As a result, the load is unevenly distributed across cores: Decoder 1 spends the longest time, while Decoder 3 sits idle for almost half the runtime.

A more intelligent solution takes advantage of tile size and complexity, which are known at encode time. We encode this data into the video bitstream as metadata, to be processed by a scheduler during decoding. Knowing the complexity of tiles lets the scheduler more evenly distribute tiles to decoder cores. To meet this goal, mVDO places a simple scheduler in front of the decoder core, as shown in Figure 6.6. This scheduler augments the tile metadata bitstream with the bitrate for each tile, as an approximation of the decode complexity. Based on this information, the scheduler sorts the tile indices, and uses that order to distribute tiles to cores.

The scheduler uses a fast bitonic sort to sort the tiles in $O(log^2(n))$ time, or about 25 cycles for 128 tiles. Based on the spread of bitrates and tiling configurations in our benchmark workloads, we determined that the sorter needs to support a bitrate range from 1–~100,000 kbps, or a 16-bit range. This means that mVDO does not correctly



(a) mVDO energy consumption and speedup, sweeping across 2000 multi-core, heterogeneous frequency configurations.

(b) Energy consumption and speedup of mVDO, now filtered to only show results under 50 mW.

Figure 6.10: We swept over heterogeneous parallel configurations for our video decoder, varying the number of cores from 4 - 128 and the number of fast and slow cores for each core count. We find a Pareto-frontier of optimal energy-speedup configurations (a). After filtering based on power consumption (b), the spread of optimal energy-speedup configurations shifts. The best Pareto-optimal point under 50 mW is a 52-core design.

sort tiles below 1 kbps or 100 Mbps. For the highest resolution video we evaluated, partitioned into 128 tiles, the metadata consumed < 300 bytes.

6.3.2 *Exploring performance impacts of scaling parallel core count*

Using the approach described above, mVDO efficiently balanced work across decoder cores. We next assessed how mVDO decoded tiled videos. As our workload, we characterized decode performance on a 4K video partitioned into 128 tiles. We scaled the hypothetical number of cores from 4 to 128 and evaluated simulated runtime and decode energy per frame.

Figure 6.9 shows how decode performance scales with the number of cores. Compared to a theoretically optimal speedup, in-order decoding was consistently 2–3× slower, due to decode dependencies constraints from row-order decoding. The Parallel-Naive design broke row dependencies by decoding tiles, but it maintained the naive scheduling pattern. On average, this design outperformed in-order decoding, but poor load distribution degraded performance as the number of cores grew to 128. The Parallel-Sched design added mVDO's optimal scheduler algorithm to the parallel cores. Even with the overhead of sorting the tile bitstream, it achieved near-optimal performance at low numbers of cores and the closest-to-optimal performance as the number of cores grew significantly.

6.3.3 *Optimizing decode latency with scalable-frequency cores*

To further optimize resource distribution on a chip with decoder units, mVDO runs cores in multiple frequency domains to enable heterogeneous decode performance

Table 6.1: mVDO Evaluation Design Points and Configuration Characteristics

Configuration	Base-Opt	Low-Power	Pareto
# Cores (Big - Little)	4 (2 - 2)	8 (2 - 6)	52 (2 - 50)
Energy / Frame (mJ)	0.606	0.56	0.52
Speedup	2.83×	4.41×	21.8×
Power (mW)	7.41	10.68	49.1

across cores. By operating at different frequencies, these heterogeneous cores can support different decode rates for tiles with varying bitrates or encoding complexity. As described in Section 6.2, VR videos can be encoded at a range of tiling configurations, which strain a parallel video in different ways. At one extreme, videos with tiled foveation may have 20-60 low-complexity tiles for peripheral regions and a smaller number of higher-complexity tiles for foveal regions. At the other extreme, a multi-layer foveated video uses only two tiles: one high-complexity tile for the foveal region, and one low-complexity tile servicing the entire peripheral region. Leveraging multiple frequency domains for mVDO's decode cores allows us to explore configurations of faster, energy-consuming cores for high-resolution areas and energy-efficient cores for low-resolution regions to optimally balance energy and delay.

To explore parallel designs with partitioned frequency domains, we swept the number of parallel cores and partitioned the cores between low- and high-frequency operating regions. The low frequency was selected to be the lowest frequency that could sustain 30 FPS throughput of a video tile, as that was the frame rate for the majority of tested videos. The high frequency was constrained the be the highest operating frequency for the decoder core. Based on the video decoder architecture we simulated, this frequency ranged from 8 - 20 MHz. To evaluate energy consumption, we calculated the energy needed to decode a synthetic video sequence engineered to represent the average of our video benchmarks in video frame complexity and CTU sizes. We computed the energy for this synthetic video and divided by the number of frames in order to report the energy consumed per frame in the synthetic video, and we evaluated across all video decoder configurations generated.

Figure 6.10 shows the results. In Figure 6.10a, we consider all architectural design points between 4 and 128 cores and all partitions of these cores across the high- and low-frequency domains; we did not eliminate any data points. We found a Pareto-frontier of optimal energy-speedup configurations that allowed for 60-100× decode latency improvement, typically requiring over 100 cores.

We used this information to select target configurations on which to evaluate mVDO performance. Keeping in mind our VR target hardware, we constrained the design space to only those designs that were under a 50 mW power budget [179]. This resulted in the configurations shown in Figure 6.10b. Though more sparse, the design space still showed performance-optimal points.

From these, we selected: (1) a Pareto-optimal design point, representing the fastest point with low energy; (2) a low-power design point, demonstrating the highest speedup

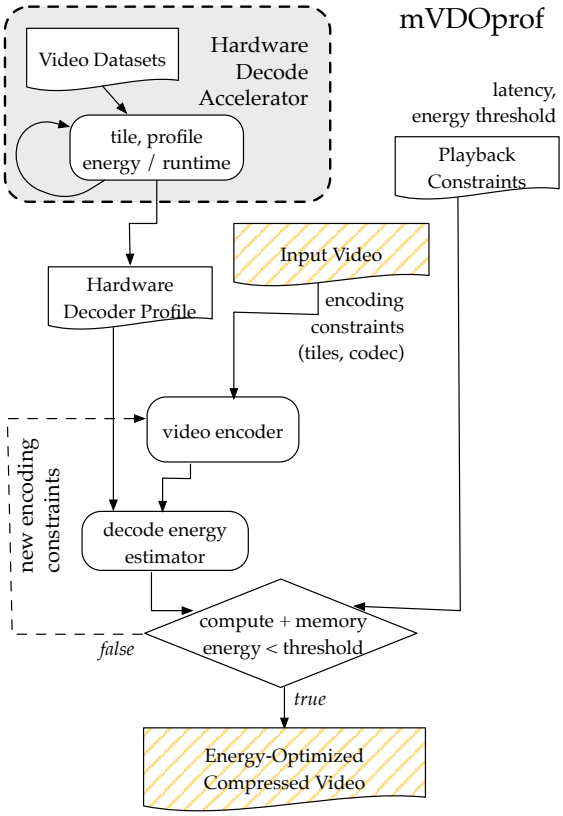


Figure 6.11: The mVDOprof framework flow helps VR developers optimize energy consumption of their videos for hardware decoders. mVDOprof generates energy profiles for video decode architectures and suggests energy-optimal tile configurations for playback constraints.

while maintaining low power consumption; and (3) a baseline-optimized point, which was the dual-frequency configuration with the best energy-speedup using 4 cores, as in the baseline architecture. Table 6.1 provides more performance details on these design points.

6.4 VIDEO ENERGY ESTIMATES WITH THE MVDOPROF FRAMEWORK

While new video decode architectures like mVDO can improve energy efficiency for video playback in constrained settings, evaluating this improvement across a range of different video encoding parameters, resolutions, and bitrates is challenging. Moreover, without access to detailed specifications and hardware designs, comparing performance across new and existing video decode architectures presents challenges. To resolve these issues, we design mVDOprof, our profiling and simulation framework for estimating energy consumption for perceptually compressed videos. mVDOprof targets mobile video decoder architectures, abstracting codec-specific complexity so creators of perceptual video compression software can evaluate the energy efficiency of their

encoded videos. We use mVDOprof in this paper to characterize energy-efficiency of videos using mVDO's simulated decoder architecture. mVDOprof can also characterize video energy on other existing architectures.

6.4.1 *mVDOprof Overview*

mVDOprof's characterizes video decode energy and latency, taking into account new perceptual compression methods to optimize energy consumption. Current approaches to these characterizations involve running exhaustive experiments on real-world hardware [106] or hand-tuning a simulator based on research prototypes and validating against full-system traces [197]. mVDOprof automates and codifies this process, decoupling the relationship between hardware decoders, input videos, and playback constraints. This enables more efficient characterization of video decode performance across more decode devices, yielding more energy-efficient VR video applications. Figure 6.11 shows a high-level mVDOprof architecture.

6.4.2 *Hardware Decoder Profiles*

Given a new hardware decoder, mVDOprof first automates profiling on large video workloads to construct a model of decode performance, the *hardware decoder profile*. This profile collects information about how a decoder performs on video benchmarks in terms of power and latency to help characterize its performance on new videos. It contains configuration and statistics files for mVDOprof and can be distributed or shared to facilitate analyzing video decode performance without access to the physical accelerator.

mVDOprof also provides an interface to construct a model from provided parameters for research prototypes or simulated chips. Hardware vendors could provide mVDOprof decoder profiles for validation, or consumer VR video application developers could build profiles for each hardware decoder their application supports.

To generate a hardware decoder profile from an existing design, the developer would run mVDOprof on the device with a power-monitoring interface enabled. For instance, to characterize the hardware decoder provided with the NVIDIA Jetson TX2 board, mVDOprof would use the on-board power monitor, to capture power consumption and latency on the video benchmarks. mVDOprof would also instrument the characterization video datasets using FFmpeg[54] and extract video statistics to profile decode events from existing data. mVDOprof would log accelerator performance on videos varying in the complexity and size of video tiles, to assess parallelism and energy efficiency. mVDOprof would then link this information to characteristics of the test videos, like number of CTUs and CTU complexity in the tile. From these logs, mVDOprof would construct a hardware decoder profile. A developer could then use this extrapolated hardware decoder profile to estimate latency and power consumption of new videos without requiring access to the physical accelerator.

mVDOprof also supports generating profiles on integrated or next-generation hardware where capturing power numbers *in situ* is challenging. For instance, to profile the hardware decoder on a Google Pixel 2 phone, the battery status monitor could be polled as a proxy for power consumption. For devices without a physical interface, e.g., next-generation decode accelerators or experimental prototypes, mVDOprof could generate a decoder profile from a given specification or simulator that provides per-video power and latency numbers.

6.4.3 *Video Encoding Constraints*

mVDOprof provides optimal encoding parameters to support energy efficiency for a VR video application. Some applications pre-determine or constrain the number of tiles used (to optimize video quality, or for integration with other performance optimizations) or mandate a specific codec for compatibility. VR video applications can specify the type of perceptual compression, such as a foveation pattern, to be explored in the simulation process. mVDOprof ingests these constraints and uses it to guide energy consumption estimates.

6.4.4 *Device Playback Constraints*

mVDOprof can optimize for a specific display device target resolution, frame rate, or potential battery life. For instance, a developer seeking to optimize video encoding to ensure the entirety of the video plays within a device's battery life could specify it via playback constraints. If playback constraints are not set, mVDOprof could suggest reducing playback parameters, i.e., resolution, to reduce energy consumption during playback.

6.4.5 *Decode Energy Estimation*

mVDOprof uses the hardware decoder profile to estimate decode energy for a specific video. This process involves using the same instrumentation of FFmpeg described in Section 6.4.2, but this time on a new video without power consumption details. mVDOprof uses FFmpeg's trace of the new video's structure along with the profile's logs for previous videos to estimate decode energy for the new video. Coupled with playback constraints, it can guide the encoding process to best optimize video decode energy consumption for a target architecture.

6.4.6 *Implementation & Methodology*

mVDOprof generated the results for our evaluation using hardware decoder profiles for each of the mVDO configurations. For mVDO's custom scheduler hardware unit, we characterized performance and energy overhead from post-synthesis results from an RTL configuration.

HARDWARE ACCELERATOR DESIGN DETAILS We constructed a baseline video decoder simulator from [180] to emulate a scalable parallel design. We implemented the mVDO decode scheduler in RTL and synthesize the designs with Yosys [189]. Our scheduler design targeted maximally supporting 128 parallel tiles. Mapped to the Nangate FreePDK 45nm library, our scheduler incurred $.035mm^2$ and sustained a clock rate of 3.4 GHz, much more than required to maintain the 100 MHz peak clock rate of the baseline architecture.

WORKLOADS mVDOprof uses two video characterization workloads to evaluate decode characteristics: `vbench`, a representative benchmark dataset from YouTube [116], and `vr-360`, a representative benchmark dataset of 4K-resolution, 360° video [38].

Table 6.2: Video Workloads Evaluated

Workload	`vbench` [22]	`VR-360` [6]
Description	YouTube	4K-360° VR
# Videos	15	9
Bitrate (Mbps)	0.53-470	10-21
Pixels / Frame	410K - 8M	8M

We used the videos at their original encoded bitrate and constructed foveated tiled and multi-resolution patterns to encode them for VR consumption. For both compression techniques, we assumed a fixed, center-weighted foveation pattern and generated videos accordingly. The foveated tile formation was uniformly tiled, with one center tile with maximum quality and concentric rings of tiles around the center tile at 75%, and 50%. We filled the rest of the frame with 10% bitrate tiles. The multi-resolution foveation pattern used two tiles: one for the entire video at 10% bitrate, and another cropped to just the central foveal region at the original bitrate. These tiles were generated and encoded by FFmpeg, invoked by mVDOprof.

6.5 EXPERIMENTAL EVALUATION

We now evaluate mVDO's performance on video characterization workloads, using mVDOprof to generate results. We first describe the energy overhead and power dedicated to mVDO at three design points. We then examine mVDO's energy consumption for regularly tiled videos, foveated tiled videos, and multi-resolution foveated videos.

6.5.1 *mVDO Overhead*

We first compare the energy consumption of mVDO's design with the baseline architecture [179]. Figure 6.12 shows the distribution of energy consumption by different aspects of the memory system when decoding a baseline 1920×1080 video.

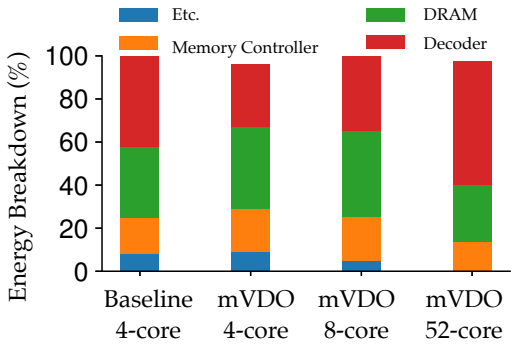


Figure 6.12: Energy breakdown of mVDO components.

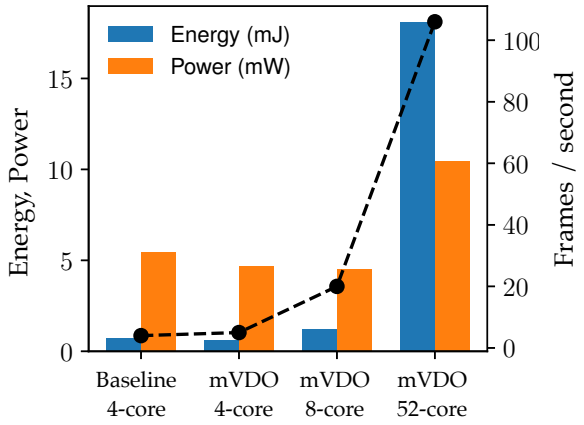


Figure 6.13: Energy consumption and total power consumption on a single video. Our 4-, 8-, and 52-core designs improve decode latency by .25 - 25× while keeping power under 15 mW.

Even with the overhead from mVDO's hardware scheduler, mVDO shows a decrease in decode energy consumption compared to the baseline. We attribute this efficiency improvement to mVDO's ability to leverage parallelism with no inter-line dependencies. Since mVDO does not require line buffers to interface between cores, its per-core energy consumption decreases by 8%. In terms of area, our scheduler takes up ~ 1% of the area, and incurs only a few cycles of latency at the initiation of the decode pipeline.

6.5.2 *Energy Efficiency*

To characterize mVDO's energy efficiency, we evaluated on our three mVDO design points: Base-Opt, Low-Power, and Pareto. For these designs, we characterized energy-efficiency in three video compression settings: (1) regularly distributed tiles, where the video was partitioned into 128 tiles of equal resolution and complexity, (2) irregular foveated tiles, where the video was partitioned into 128 tiles in a foveated pattern, and

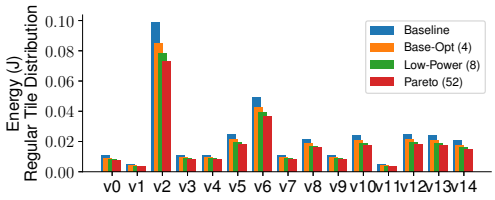
(3) multi-resolution foveation, where the video was partitioned into two regions, foveal and peripheral. We report energy consumption for the full video decoder chip.

REGULARLY TILED VIDEO. Figure 6.14a and Figure 6.14b show the results of mVDO's architecture on a regularly tiled workload. Compared to the 4-core design, the 8- and 52-core designs provided a 6-15% energy-efficiency benefit. This benefit is more pronounced for the larger, high-resolution videos in `vr-360`. At smaller video sizes (e.g., vbench's $v1$ video) this energy-efficiency benefit is negligible. This indicates that for small-display, resource-constrained video playback, a small number of cores can provide sufficient energy efficiency, especially for traditionally encoded video. The speedup and potential energy-efficiency gains provided by the Pareto or even Low-Power design point did not significantly reduce energy consumption.

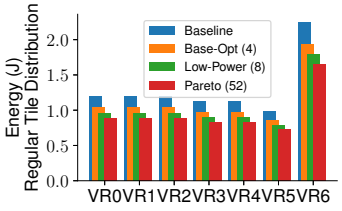
TILED, FOVEATED VIDEO. For tiled, foveated video, however, the results differed. For this workload, energy consumption dropped significantly across the three designs This demonstrates how our heterogeneous parallel cores reduced energy consumption. Figure 6.14c and Figure 6.14d demonstrate the energy consumed across all videos.

We observe that energy consumption was particularly sensitive to the distribution of tiles across the two frequency domains. mVDO's scheduler found an optimal balance between the two cores, but a poor allocation of tiles to decoder cores results in much higher energy consumption for the three mVDO designs. While even poor tile decode scheduling can demonstrate scalability on our designs, with energy consumption decreasing as we add more cores, the change in energy consumption was not as significant. With the optimal decode schedule, the Low-Power design reduced energy consumption by a factor of 2, and the Pareto-optimal design achieved almost a 4× energy consumption reduction. Though this performance did not scale linearly, it was much more scalable than the baseline at 1-core.

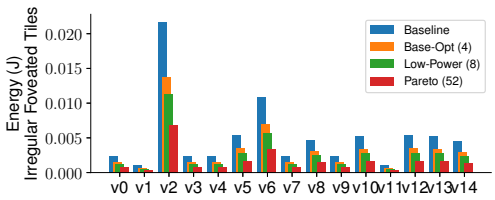
MULTI-RESOLUTION VIDEO. Multi-resolution is the most extreme video compression method. The only effective decode schedule available is for mVDO's scheduler to map the high-resolution foveal tile to the big core and the low-resolution peripheral tile to the little core. This highlights a limitation in mVDO's scheduling design: distinct decode cores cannot reclaim work from a tile and instead sit idle. Figure 6.14e and Figure 6.14f graph energy consumption for the three designs, which showed similar trends. Like the regularly-tiled use case, the multi-resolution approach demonstrated a small energy-efficiency benefit, 6–15%. However, the energy consumed to decode a multi-resolution video was 200× lower than that consumed for regularly-tiled video, highlighting the benefits of using perceptually compressed video.



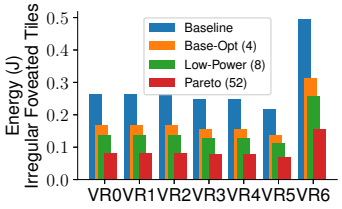
(a) vbench energy consumption results for regularly distributed tiles.



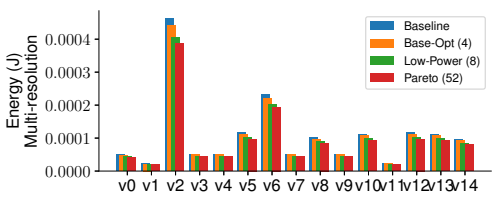
(b) `vr-360` energy consumption results for regularly distributed tiles.



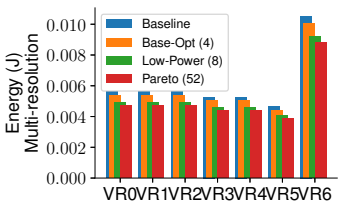
(c) vbench energy consumption results for irregular foveated tiles.



(d) `vr-360` energy consumption results for irregular foveated tiles.



(e) vbench energy consumption results for multi-resolution foveation.



(f) `vr-360` energy consumption results for multi-resolution foveation.

Figure 6.14: Energy consumption for decoding vbench and `vr-360` videos. vbench videos range in resolution and show a commensurate range in energy consumption. mVDO reduces energy for all compression styles but is most scalable for foveated tiles and lowest-energy for multi-resolution foveation. `vr-360` show greater benefit from mVDO's scalability, indicating that larger, high-resolution VR content benefits more from mVDO's optimizations.

6.6 SUMMARY

Emerging techniques for perceptually compressing videos provide network bandwidth benefits, but video decoder architectures have not innovated alongside them. To adapt VR video decoder architectures with energy efficiency in mind, this paper proposed mVDO, a new parallel accelerator optimized for VR video decompression. mVDO's design uses heterogeneous parallel cores to minimize decode latency and distribute decode effort efficiently, reducing energy by an average of 12–22% per frame. Moreover, mVDO can efficiently decode perceptually compressed videos to achieve 4.5× better energy efficiency over traditionally compressed video. Our new tool, mVDOprof, helps VR developers characterize the energy consumption of emerging perceptually-compressed video techniques on video decoder hardware. mVDOprof can help VR software developers and ASIC designers better tune their workloads and optimize energy consumption as the VR software and hardware ecosystem continues to evolve. As performance demands for video encoding and processing techniques grow more stringent, mVDO and mVDOprof help to close the gap for optimizing the energy consumption of future video workloads.

7

CONCLUSION

In this thesis, I described a series of systems for applying perceptual information to video capture, processing and storage systems. The computer systems and architectures targeted by the work in this dissertation span software and hardware design, and leverage application insights and codesign principles to fully leverage perceptual optimizations. I described hardware-software codesign work for mobile vision on energy-harvesting cameras, real-time virtual reality video synthesis, and energy-efficient mobile video playback. For cloud-scale video systems, I demonstrated how saliency can be used to improve video storage and processing.

Video data continues to grow with increased video capture and consumption trends. Leveraging perceptual cues can help manage this data. By abstracting per-pixel information into a perceptual layer, we can optimize computer system performance based on an image's visual structure, semantics, or saliency. My thesis presents several examples of applying perceptual cues to modern visual computing tasks to achieve faster or real-time performance, improved energy efficiency, or smaller data bandwidth.

Looking forward, visual media will undoubtedly continue to drive use of computational resources. Vision and computational photography algorithms will continue to improve. Demand for richer displays and cameras will continue to grow. In response, compute infrastructure will become increasingly more specialized. While traditional systems concerns like performance and energy efficiency will drive new system and architecture designs, systems will also make use of more perceptual information to deliver improved performance. Moreover, computer systems will need to optimize not only for *intrinsic* perceptual cues in visual media, but also perceptual characteristics of the end-application. Visual computing workloads for human viewing consumption, like HFBS, Vignette and mVDO, highlight different optimization criteria from than workloads meant for machine consumption, like the near-sensor processor. As application-specific systems bifurcate into optimized workloads for algorithmic consumption or end-users, this thesis on perceptually-optimized visual computing workloads and architectures serves as a launch point for future visual computing architectures.

BIBLIOGRAPHY

[1] Anne Aaron, Zhi Li, Megha Manohara, Jan De Cock, and David Ronca. *Per-Title Encode Optimization*. Tech. rep. Accessed: 2018-06-08. 2015.

[2] Anne Aaron and David Ronca. *High Quality Video Encoding at Scale*. Tech. rep. Accessed: 2018-06-08. 2015.

[3] Andrew Adams, Jongmin Baek, and Myers Abraham Davis. "Fast high-dimensional filtering using the permutohedral lattice." In: *Eurographics* (2010).

[4] Andrew Adams, Eino-Ville Talvala, Sung Hee Park, David E Jacobs, Boris Ajdin, Natasha Gelfand, Jennifer Dolson, Daniel Vaquero, Jongmin Baek, Marius Tico, et al. "The Frankencamera: an experimental platform for computational photography." In: *ACM Transactions on Graphics (TOG)* 29.4 (2010), p. 29.

[5] A. Alaghi, Cheng Li, and J.P. Hayes. "Stochastic circuits for real-time image-processing applications." In: *Design Automation Conference (DAC), 2013 50th ACM/EDAC/IEEE*. May 2013.

[6] Rachel Albert, Anjul Patney, David Luebke, and Joohwan Kim. "Latency Requirements for Foveated Rendering in Virtual Reality." In: *ACM Trans. Appl. Percept.* 14.4 (Sept. 2017), 25:1–25:13. ISSN: 1544-3558. DOI: 10.1145/3127589. URL: http://doi.acm.org/10.1145/3127589.

[7] Brandon Amos, Bartosz Ludwiczuk, and Mahadev Satyanarayanan. *OpenFace: A general-purpose face recognition library with mobile applications*. Tech. rep. CMU-CS-16-118, CMU School of Computer Science, 2016.

[8] Michael R Anderson, Michael Cafarella, German Ros, and Thomas F Wenisch. "Physical representation-based predicate optimization for a visual analytics database." In: *ICDE* (2019).

[9] Robert Anderson, David Gallup, Jonathan T Barron, Janne Kontkanen, Noah Snavely, Carlos Hernández, Sameer Agarwal, and Steven M Seitz. "Jump: Virtual Reality Video." In: *SIGGRAPH Asia* (2016).

[10] Alliance for Open Media (AOM). *AV1 Software Repository*. https://aomedia.googlesource.com/aom. 2018.

[11] Kenneth C. Barr and Krste Asanović. "Energy-aware Lossless Data Compression." In: *ACM Trans. Comput. Syst.* 24.3 (Aug. 2006), pp. 250–291. ISSN: 0734-2071. DOI: 10.1145/1151690.1151692. URL: http://doi.acm.org/10.1145/1151690.1151692.

[12] Jonathan T Barron, Andrew Adams, YiChang Shih, and Carlos Hernández. "Fast bilateral-space stereo for synthetic defocus." In: *CVPR* (2015).

[13] Jonathan T Barron and Ben Poole. "The fast bilateral solver." In: *ECCV* (2016).

[14] Brendan Barry, Cormac Brick, Fergal Connor, David Donohoe, David Moloney, Richard Richmond, Martin O'Riordan, and Vasile Toma. "Always-on Vision Processing Unit for Mobile Applications." In: *IEEE Micro* 35.2 (2015), pp. 56–66.

[15] Behnam Bastani and Eric Turner. *Google AI Blog*. Dec. 2017. URL: https://ai.googleblog.com/2017/12/introducing-new-foveation-pipeline-for.html.

[16] Doug Beaver, Sanjeev Kumar, Harry Li, Jason Sobel, and Peter Vajgel. "Finding a needle in Haystack: Facebook's photo storage." In: *USENIX Symposium on Operating Systems Design and Implementation (OSDI)*. USENIX Association, 2010. URL: https://www.usenix.org/legacy/event/osdi10/.

[17] Michelle A Borkin, Azalea A Vo, Zoya Bylinskii, Phillip Isola, Shashank Sunkavalli, Aude Oliva, and Hanspeter Pfister. "What makes a visualization memorable?" In: *IEEE Transactions on Visualization and Computer Graphics* 19.12 (2013), pp. 2306–2315.

[18] Amirali Boroumand et al. "Google Workloads for Consumer Devices: Mitigating Data Movement Bottlenecks." In: *Proceedings of the Twenty-Third International Conference on Architectural Support for Programming Languages and Operating Systems*. ASPLOS '18. Williamsburg, VA, USA: ACM, 2018, pp. 316–331. ISBN: 978-1-4503-4911-6. DOI: 10.1145/3173162.3173177. URL: http://doi.acm.org/10.1145/3173162.3173177.

[19] Mark Buckler, Philip Bedoukian, Suren Jayasuriya, and Adrian Sampson. "EVA^2: Exploiting Temporal Redundancy in Live Computer Vision." In: *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*. IEEE. 2018, pp. 533–546.

[20] Andreas Bulling, Daniel Roggen, and Gerhard Tröster. "Wearable EOG Goggles: Seamless Sensing and Context-awareness in Everyday Environments." In: *J. Ambient Intell. Smart Environ.* 1.2 (Apr. 2009), pp. 157–171. ISSN: 1876-1364. URL: http://dl.acm.org/citation.cfm?id=2350315.2350320.

[21] Zoya Bylinskii, Nam Wook Kim, Peter O'Donovan, Sami Alsheikh, Spandan Madan, Hanspeter Pfister, Fredo Durand, Bryan Russell, and Aaron Hertzmann. "Learning Visual Importance for Graphic Designs and Data Visualizations." In: *Proceedings of the 30th Annual ACM Symposium on User Interface Software and Technology*. UIST '17. Québec City, QC, Canada: ACM, 2017, pp. 57–69. ISBN: 978-1-4503-4981-9. DOI: 10.1145/3126594.3126653. URL: http://doi.acm.org/10.1145/3126594.3126653.

[22] Zoya Bylinskii, Adrià Recasens, Ali Borji, Aude Oliva, Antonio Torralba, and Frédo Durand. "Where Should Saliency Models Look Next?" In: *European Conference on Computer Vision (ECCV)*. Ed. by Bastian Leibe, Jiri Matas, Nicu Sebe, and Max Welling. Springer International Publishing, 2016.

[23] Joel Carranza, Christian Theobalt, Marcus A Magnor, and Hans-Peter Seidel. "Free-viewpoint video of human actors." In: 3 (2003).

[24] Chad Carson, Megan Thomas, Serge Belongie, Joseph M Hellerstein, and Jitendra Malik. "Blobworld: A system for region-based image indexing and retrieval." In: *International conference on advances in visual information systems*. Springer. 1999, pp. 509–517.

[25] Centeye. *Vision Chips*. http://www.centeye.com/technology/vision-chips/. Accessed: 2017-06-06.

[26] Derek Chan, Hylke Buisman, Christian Theobalt, and Sebastian Thrun. "A noise-aware filter for real-time depth upsampling." In: *ECCV Workshops* (2008).

[27] N. Chandramoorthy, G. Tagliavini, K. Irick, A. Pullini, S. Advani, S. A. Habsi, M. Cotter, J. Sampson, V. Narayanan, and L. Benini. "Exploring architectural heterogeneity in intelligent vision systems." In: *2015 IEEE 21st International Symposium on High Performance Computer Architecture (HPCA)*. Feb. 2015, pp. 1–12. DOI: 10.1109/HPCA.2015.7056017.

[28] Surajit Chaudhuri and Luis Gravano. "Optimizing queries over multimedia repositories." In: *ACM SIGMOD Record*. Vol. 25. 2. ACM. 1996, pp. 91–102.

[29] Huaijin G. Chen, Suren Jayasuriya, Jiyue Yang, Judy Stephen, Sriram Sivaramakrishnan, Ashok Veeraraghavan, and Alyosha Molnar. "ASP Vision: Optically Computing the First Layer of Convolutional Neural Networks Using Angle Sensitive Pixels." In: *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2016.

[30] Jiawen Chen, Sylvain Paris, and Frédo Durand. "Real-time edge-aware image processing with the bilateral grid." In: *SIGGRAPH* (2007).

[31] Chen, Yu-Hsin and Krishna, Tushar and Emer, Joel and Sze, Vivienne. "Eyeriss: An Energy-Efficient Reconfigurable Accelerator for Deep Convolutional Neural Networks." In: *IEEE International Solid-State Circuits Conference, ISSCC 2016, Digest of Technical Papers*. 2016, 262–263.

[32] Junguk Cho, B. Benson, S. Mirzaei, and R. Kastner. "Parallelized Architecture of Multiple Classifiers for Face Detection." In: *20th IEEE International Conference on Application-specific Systems, Architectures and Processors, 2009. ASAP 2009*. July 2009, pp. 75–82. DOI: 10.1109/ASAP.2009.38.

[33] J. Choi, S. Park, J. Cho, and E. Yoon. "A 3.4 μW CMOS image sensor with embedded feature-extraction algorithm for motion-triggered object-of-interest imaging." In: *2013 IEEE International Solid-State Circuits Conference Digest of Technical Papers*. Feb. 2013, pp. 478–479. DOI: 10.1109/ISSCC.2013.6487822.

[34] François Chollet et al. *Keras*. https://keras.io. 2015.

[35] Cisco. *Cisco Visual Networking Index: Forecast and Methodology, 2008–2013*. Tech. rep. Accessed: 2018-06-07. 2008.

[36] Jason Clemons, Chih-Chi Cheng, Iuri Frosio, Daniel Johnson, and Stephen W. Keckler. "A Patch Memory System For Image Processing and Computer Vision." In: *2016 49th IEEE/ACM International Symposium on Microarchitecture, 2016. MICRO-49*. Oct. 2016.

[37] Jason Clemons, Andrea Pellegrini, Silvio Savarese, and Todd Austin. "EVA: An Efficient Vision Architecture for Mobile Systems." In: *Proceedings of the 2013 International Conference on Compilers, Architectures and Synthesis for Embedded Systems*. Montreal, Quebec, Canada: IEEE Press, 2013, 13:1–13:10. ISBN: 978-1-4799-1400-5.

[38] Xavier Corbillon, Francesca De Simone, and Gwendal Simon. "360Degreee Video Head Movement Dataset." In: *Proceedings of the 8th ACM on Multimedia Systems Conference*. MMSys'17. Taipei, Taiwan: ACM, 2017, pp. 199–204. ISBN: 978-1-4503-5002-0. DOI: 10.1145/3083187.3083215. URL: http://doi.acm.org/10.1145/3083187.3083215.

[39] Marcella Cornia, Lorenzo Baraldi, Giuseppe Serra, and Rita Cucchiara. "A Deep Multi-Level Network for Saliency Prediction." In: *International Conference on Pattern Recognition (ICPR)*. 2016.

[40] Eduardo Cuervo, Aruna Balasubramanian, Dae-ki Cho, Alec Wolman, Stefan Saroiu, Ranveer Chandra, and Paramvir Bahl. "MAUI: making smartphones last longer with code offload." In: *Proceedings of the 8th international conference on Mobile systems, applications, and services*. ACM. 2010, pp. 49–62.

[41] Jan De Cock, Aditya Mavlankar, Anush Moorthy, and Anne Aaron. "A large-scale video codec comparison of x264, x265 and libvpx for practical VOD applications." In: *Applications of Digital Image Processing XXXIX*. Vol. 9971. International Society for Optics and Photonics. 2016, p. 997116.

[42] Zidong Du, Robert Fasthuber, Tianshi Chen, Paolo Ienne, Ling Li, Tao Luo, Xiaobing Feng, Yunji Chen, and Olivier Temam. "ShiDianNao: Shifting Vision Processing Closer to the Sensor." In: *Proceedings of the 42Nd Annual International Symposium on Computer Architecture*. ISCA '15. New York, NY, USA: ACM, 2015, pp. 92–104. ISBN: 978-1-4503-3402-0. DOI: 10.1145/2749469.2750389. (Visited on 09/29/2015).

[43] H. Esmaeilzadeh, P. Saeedi, B.N. Araabi, C. Lucas, and Sied Mehdi Fakhraie. "Neural network stream processing core (NnSP) for embedded systems." In: *Circuits and Systems, 2006. ISCAS 2006. Proceedings. 2006 IEEE International Symposium on*. May 2006. DOI: 10.1109/ISCAS.2006.1693199.

[44] Hadi Esmaeilzadeh, Emily Blem, Renee St Amant, Karthikeyan Sankaralingam, and Doug Burger. "Dark silicon and the end of multicore scaling." In: *2011 38th Annual international symposium on computer architecture (ISCA)*. IEEE. 2011, pp. 365–376.

[45] Facebook. *Facebook Surround 360*. Accessed: 2017-06-15. 2017.

[46] Christos Faloutsos, Ron Barber, Myron Flickner, Jim Hafner, Wayne Niblack, Dragutin Petkovic, and William Equitz. "Efficient and effective querying by image content." In: *Journal of intelligent information systems* 3.3-4 (1994), pp. 231–262.

[47] Ching-Ling Fan, Jean Lee, Wen-Chih Lo, Chun-Ying Huang, Kuan-Ta Chen, and Cheng-Hsin Hsu. "Fixation Prediction for 360&Deg; Video Streaming in Head-Mounted Virtual Reality." In: *Proceedings of the 27th Workshop on Network and Operating Systems Support for Digital Audio and Video*. NOSSDAV'17. Taipei, Taiwan: ACM, 2017, pp. 67–72. ISBN: 978-1-4503-5003-7. DOI: 10.1145/3083165.3083180. URL: http://doi.acm.org/10.1145/3083165.3083180.

[48] C. Farabet, B. Martini, P. Akselrod, S. Talay, Y. LeCun, and E. Culurciello. "Hardware accelerated convolutional neural networks for synthetic vision systems." In: *Circuits and Systems (ISCAS), Proceedings of 2010 IEEE International Symposium on*. May 2010, pp. 257–260. DOI: 10.1109/ISCAS.2010.5537908.

[49] C. Farabet, B. Martini, B. Corda, P. Akselrod, E. Culurciello, and Y. LeCun. "NeuFlow: A runtime reconfigurable dataflow processor for vision." In: *2011 IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*. June 2011. DOI: 10.1109/CVPRW.2011.5981829.

[50] Zeev Farbman, Raanan Fattal, Dani Lischinski, and Richard Szeliski. "Edge-preserving decompositions for multi-scale tone and detail manipulation." In: *SIGGRAPH* (2008).

[51] *Facebook Live | Live Video Streaming*. https://live.fb.com/. 2018.

[52] Travis Felder. "Teaching With NYT Virtual Reality Across Subjects." In: *New York Times* (Mar. 2019). URL: https://www.nytimes.com/2019/03/28/learning/lesson-plans/teaching-with-nyt-virtual-reality-across-subjects.html.

[53] David Ferstl, Christian Reinbacher, Rene Ranftl, Matthias Ruether, and Horst Bischof. "Image guided depth upsampling using anisotropic total generalized variation." In: *ICCV* (2013).

[54] Fabrice Bellard. *FFmpeg*. https://ffmpeg.org.

[55] D. Fick, G. Kim, A. Wang, D. Blaauw, and D. Sylvester. "Mixed-signal stochastic computation demonstrated in an image sensor with integrated 2D edge detection and noise filtering." In: *Proceedings of the IEEE 2014 Custom Integrated Circuits Conference*. Sept. 2014, pp. 1–4. DOI: 10.1109/CICC.2014.6946130.

[56] D. F. Finchelstein, V. Sze, and A. P. Chandrakasan. "Multicore Processing and Efficient On-Chip Caching for H.264 and Future Video Decoders." In: *IEEE Transactions on Circuits and Systems for Video Technology* 19.11 (Nov. 2009), pp. 1704–1713. ISSN: 1558-2205. DOI: 10.1109/TCSVT.2009.2031459.

[57] R. Forchheimer and A. Astrom. "Near-sensor image processing: a new paradigm." In: *Image Processing, IEEE Transactions on* 3.6 (Nov. 1994), pp. 736–746. ISSN: 1057-7149. DOI: 10.1109/83.336244.

[58] Sadjad Fouladi, John Emmons, Emre Orbay, Catherine Wu, Riad S. Wahby, and Keith Winstein. "Salsify: Low-Latency Network Video through Tighter Integration between a Video Codec and a Transport Protocol." In: *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*. Renton, WA: USENIX Association, 2018, pp. 267–282. ISBN: 978-1-931971-43-0. URL: https://www.usenix.org/conference/nsdi18/presentation/fouladi.

[59] Sadjad Fouladi, Riad S. Wahby, Brennan Shacklett, Karthikeyan Vasuki Balasubramaniam, William Zeng, Rahul Bhalerao, Anirudh Sivaraman, George Porter, and Keith Winstein. "Encoding, Fast and Slow: Low-Latency Video Processing Using Thousands of Tiny Threads." In: *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*. Boston, MA: USENIX Association, 2017, pp. 363–376. ISBN: 978-1-931971-37-9. URL: https://www.usenix.org/conference/nsdi17/technical-sessions/presentation/fouladi.

[60] Blender Foundation. *Blender Open Projects*. Tech. rep. Accessed: 2018-07-11. 2002.

[61] Yoav Freund and Robert E. Schapire. *A Decision-Theoretic Generalization of on-Line Learning and an Application to Boosting*. 1997.

[62] Eduardo S. L. Gastal and Manuel M. Oliveira. "Domain transform for edge-aware image and video processing." In: *SIGGRAPH* (2011).

[63] Wilson S Geisler and Jeffrey S Perry. "Real-time foveated multiresolution system for low-bandwidth video communication." In: *Human vision and electronic imaging III*. Vol. 3299. International Society for Optics and Photonics. 1998, pp. 294–305.

[64] Michaël Gharbi, YiChang Shih, Gaurav Chaurasia, Jonathan Ragan-Kelley, Sylvain Paris, and Frédo Durand. "Transform Recipes for Efficient Cloud Photo Enhancement." In: *ACM Trans. Graph.* 34.6 (Oct. 2015), 228:1–228:12. ISSN: 0730-0301. DOI: 10.1145/2816795.2818127. URL: http://doi.acm.org/10.1145/2816795.2818127.

[65] Google. *CardBoard - Google VR*. https://vr.google.com/cardboard/. Accessed: 2017-06-06.

[66] GoPro. *OmniTM - All Inclusive*. https://shop.gopro.com/virtualreality/omni---all-inclusive. Accessed: 2017-06-06.

[67] Brian Guenter, Mark Finch, Steven Drucker, Desney Tan, and John Snyder. "Foveated 3D graphics." In: *ACM Transactions on Graphics (TOG)* 31.6 (2012), p. 164.

[68] C. Guo and L. Zhang. "A Novel Multiresolution Spatiotemporal Saliency Detection Model and Its Applications in Image and Video Compression." In: *IEEE Transactions on Image Processing* 19.1 (Jan. 2010), pp. 185–198. ISSN: 1057-7149. DOI: 10.1109/TIP.2009.2030969.

[69] Qing Guo, Karin Strauss, Luis Ceze, and Henrique S. Malvar. "High-Density Image Storage Using Approximate Memory Cells." In: *Proceedings of the Twenty-First International Conference on Architectural Support for Programming Languages and Operating Systems*. ASPLOS '16. Atlanta, Georgia, USA: ACM, 2016, pp. 413–426. ISBN: 978-1-4503-4091-5. DOI: 10.1145/2872362.2872413. URL: http://doi.acm.org/10.1145/2872362.2872413.

[70] Rupesh Gupta, Meera Thapar Khanna, and Santanu Chaudhury. "Visual saliency guided video compression algorithm." In: *Signal Processing: Image Communication* 28.9 (2013), pp. 1006–1022. ISSN: 0923-5965. DOI: https://doi.org/10.1016/j.image.2013.07.003. URL: http://www.sciencedirect.com/science/article/pii/S0923596513000982.

[71] H. Hadizadeh and I. V. Bajić. "Saliency-Aware Video Compression." In: *IEEE Transactions on Image Processing* 23.1 (Jan. 2014), pp. 19–33. ISSN: 1057-7149. DOI: 10.1109/TIP.2013.2282897.

[72] Seungyeop Han, Rajalakshmi Nandakumar, Matthai Philipose, Arvind Krishnamurthy, and David Wetherall. "GlimpseData: Towards Continuous Vision-based Personal Analytics." In: *Proceedings of the 2014 Workshop on Physical Analytics*. Bretton Woods, New Hampshire, USA: ACM, 2014, pp. 31–36. ISBN: 978-1-4503-2825-8. DOI: 10.1145/2611264.2611269. URL: http://doi.acm.org/10.1145/2611264.2611269.

[73] Seungyeop Han and Matthai Philipose. "The Case for Onloading Continuous High-datarate Perception to the Phone." In: *Proceedings of the 14th USENIX Conference on Hot Topics in Operating Systems*. HotOS'13. Santa Ana Pueblo, New Mexcio: USENIX Association, 2013.

[74] Johann Hauswald, Thomas Manville, Qi Zheng, Ronald Dreslinski, Chaitali Chakrabarti, and Trevor Mudge. "A hybrid approach to offloading mobile image classification." In: *Acoustics, Speech and Signal Processing (ICASSP), 2014 IEEE International Conference on*. IEEE. 2014, pp. 8375–8379.

[75] Brandon Haynes, Amrita Mazumdar, Armin Alaghi, Magdalena Balazinska, Luis Ceze, and Alvin Cheung. "LightDB:A DBMS for Virtual Reality." In: *Proc. VLDB Endow.* 11.10 (June 2018).

[76] Brandon Haynes, Artem Minyaylov, Magdalena Balazinska, Luis Ceze, and Alvin Cheung. "VisualCloud Demonstration: A DBMS for Virtual Reality." In: *Proceedings of the 2017 ACM International Conference on Management of Data*. SIGMOD '17. Chicago, Illinois, USA: ACM, 2017, pp. 1615–1618. ISBN: 978-1-4503-4197-4. DOI: 10.1145/3035918.3058734. URL: http://doi.acm.org/10.1145/3035918.3058734.

[77] Jian He, Mubashir Adnan Qureshi, Lili Qiu, Jin Li, Feng Li, and Lei Han. "Rubiks: Practical 360-Degree Streaming for Smartphones." In: *Proceedings of the 16th Annual International Conference on Mobile Systems, Applications, and Services*. MobiSys '18. Munich, Germany: ACM, 2018, pp. 482–494. ISBN: 978-1-4503-5720-

3. DOI: 10.1145/3210240.3210323. URL: http://doi.acm.org/10.1145/3210240.3210323.

[78] Kaiming He and Jian Sun. "Fast guided filter." In: *CoRR* abs/1505.00996 (2015).

[79] Kaiming He, Jian Sun, and Xiaoou Tang. "Guided image filtering." In: *ECCV* (2010).

[80] Daniel Hefenbrock, Jason Oberg, Nhat Tan Nguyen Thanh, Ryan Kastner, and Scott B. Baden. "Accelerating viola-jones face detection to fpga-level using gpus." In: *2010 18th IEEE Annual International Symposium on Field-Programmable Custom Computing Machines*. IEEE, 2010. (Visited on 10/05/2015).

[81] M. Hiromoto, K. Nakahara, H. Sugano, Y. Nakamura, and R. Miyamoto. "A Specialized Processor Suitable for AdaBoost-Based Detection with Haar-like Features." In: *IEEE Conference on Computer Vision and Pattern Recognition, 2007. CVPR '07*. June 2007, pp. 1–8. DOI: 10.1109/CVPR.2007.383415.

[82] Kevin Hsieh, Ganesh Ananthanarayanan, Peter Bodik, Shivaram Venkataraman, Paramvir Bahl, Matthai Philipose, Phillip B. Gibbons, and Onur Mutlu. "Focus: Querying Large Video Datasets with Low Latency and Low Cost." In: *OSDI*. 2018. URL: https://www.usenix.org/conference/osdi18/presentation/hsieh.

[83] HTC. *HTC Vive Pro Eye*. Feb. 2020. URL: https://enterprise.vive.com/us/product/vive-pro-eye/.

[84] Fu-Chung Huang, Kevin Chen, and Gordon Wetzstein. "The light field stereoscope: immersive computer graphics via factored near-eye light field displays with focus cues." In: *ACM Transactions on Graphics (TOG)* (2015). ISSN: 0730-0301.

[85] Qi Huang et al. "SVE: Distributed Video Processing at Facebook Scale." In: *Proceedings of the 26th Symposium on Operating Systems Principles*. SOSP '17. Shanghai, China: ACM, 2017, pp. 87–103. ISBN: 978-1-4503-5085-3. DOI: 10.1145/3132747.3132775. URL: http://doi.acm.org/10.1145/3132747.3132775.

[86] X. Huang, C. Shen, X. Boix, and Q. Zhao. "SALICON: Reducing the Semantic Gap in Saliency Prediction by Adapting Deep Neural Networks." In: *2015 IEEE International Conference on Computer Vision (ICCV)*. Dec. 2015, pp. 262–270. DOI: 10.1109/ICCV.2015.38.

[87] Texas Instruments. *MSP430-GCC - Open Source Compiler for MSP430 Microcontrollers*. http://www.ti.com/tool/msp430-gcc-opensource. 2015.

[88] Texas Instruments. *OMAP*. http://processors.wiki.ti.com/index.php/OMAP. Accessed: 2017-06-06.

[89] L. Itti, C. Koch, and E. Niebur. "A model of saliency-based visual attention for rapid scene analysis." In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 20.11 (Nov. 1998), pp. 1254–1259. ISSN: 0162-8828. DOI: 10.1109/34.730558.

[90] Djordje Jevdjic, Karin Strauss, Luis Ceze, and Henrique S. Malvar. "Approximate Storage of Compressed and Encrypted Videos." In: *Proceedings of the Twenty-Second International Conference on Architectural Support for Programming Languages and Operating Systems*. ASPLOS '17. Xi'an, China: ACM, 2017, pp. 361–373. ISBN: 978-1-4503-4465-4. DOI: 10.1145/3037697.3037718. URL: http://doi.acm.org/10.1145/3037697.3037718.

[91] Junchen Jiang, Ganesh Ananthanarayanan, Peter Bodik, Siddhartha Sen, and Ion Stoica. "Chameleon: scalable adaptation of video analytics." In: *SIGCOMM*. 2018. DOI: 10.1145/3230543.3230574. URL: https://doi.org/10.1145/3230543.3230574.

[92] Y. Jing and S. Baluja. "VisualRank: Applying PageRank to Large-Scale Image Search." In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 30.11 (Nov. 2008), pp. 1877–1890. ISSN: 0162-8828. DOI: 10.1109/TPAMI.2008.121.

[93] Norman P. Jouppi et al. "In-Datacenter Performance Analysis of a Tensor Processing Unit." In: *Proceedings of the 44th Annual International Symposium on Computer Architecture*. ISCA '17. Toronto, ON, Canada: ACM, 2017, pp. 1–12. ISBN: 978-1-4503-4892-8. DOI: 10.1145/3079856.3080246. URL: http://doi.acm.org/10.1145/3079856.3080246.

[94] Daniel Kang, Peter Bailis, and Matei Zaharia. "BlazeIt: Fast Exploratory Video Queries using Neural Networks." In: *CoRR* abs/1805.01046 (2018).

[95] Daniel Kang, John Emmons, Firas Abuzaid, Peter Bailis, and Matei Zaharia. "NoScope: Optimizing Deep CNN-Based Queries over Video Streams at Scale." In: *PVLDB* 10.11 (2017), pp. 1586–1597.

[96] V. Kantorov and I. Laptev. "Efficient feature extraction, encoding and classification for action recognition." In: *Proc. Computer Vision and Pattern Recognition (CVPR), IEEE, 2014*. 2014.

[97] Anton S Kaplanyan, Anton Sochenov, Thomas Leimkühler, Mikhail Okunev, Todd Goodall, and Gizem Rufo. "DeepFovea: neural reconstruction for foveated rendering and video compression using learned statistics of natural videos." In: *ACM Transactions on Graphics (TOG)* 38.6 (2019), p. 212.

[98] Ioannis Katsavounidis. *Dynamic optimizer – a perceptual video encoding optimization framework*. Tech. rep. Accessed: 2018-06-08. 2018.

[99] Johannes Kopf, Michael F. Cohen, Dani Lischinski, and Matt Uyttendaele. "Joint bilateral upsampling." In: *SIGGRAPH* (2007).

[100] Sanjay Krishnan, Adam Dziedzic, and Aaron J. Elmore. "DeepLens: Towards a Visual Data Management System." In: *CIDR* (2018).

[101] Gameface Labs. *GameFace Labs Android-based VR Headset Powered by NVIDIA Jetson TX2*. Accessed: 2019-11-24. 2019.

[102] Z. Lai, Y. C. Hu, Y. Cui, L. Sun, N. Dai, and H. Lee. "Furion: Engineering High-Quality Immersive Virtual Reality on Today's Mobile Devices." In: *IEEE Transactions on Mobile Computing* (2019), pp. 1–1. DOI: 10.1109/TMC.2019.2913364.

[103] J. S. Lee and T. Ebrahimi. "Perceptual Video Compression: A Survey." In: *IEEE Journal of Selected Topics in Signal Processing* 6.6 (Oct. 2012), pp. 684–697. ISSN: 1932-4553. DOI: 10.1109/JSTSP.2012.2215006.

[104] V. T. Lee, A. Alaghi, and L. Ceze. "Correlation manipulating circuits for stochastic computing." In: *2018 Design, Automation Test in Europe Conference Exhibition (DATE)*. Mar. 2018, pp. 1417–1422. DOI: 10.23919/DATE.2018.8342234.

[105] Yue Leng, Chi-Chun Chen, Qiuyue Sun, Jian Huang, and Yuhao Zhu. "Semantic-Aware Virtual Reality Video Streaming." In: *Proceedings of the 9th Asia-Pacific Workshop on Systems*. ACM. 2018, p. 21.

[106] Yue Leng, Chi-Chun Chen, Qiuyue Sun, Jian Huang, and Yuhao Zhu. "Energy-efficient Video Processing for Virtual Reality." In: *Proceedings of the 46th International Symposium on Computer Architecture*. ISCA '19. Phoenix, Arizona: ACM, 2019, pp. 91–103. ISBN: 978-1-4503-6669-4. DOI: 10.1145/3307650.3322264. URL: http://doi.acm.org/10.1145/3307650.3322264.

[107] Yue Leng, Chi-chun Chen, Qiuyue Sun, Jian Huang, and Yuhao Zhu. "Energy-Efficient Video Processing for Virtual Reality." In: *ISCA*. ACM. 2019.

[108] Marc Levoy and Pat Hanrahan. "Light field rendering." In: *SIGGRAPH* (1996).

[109] Zhi Li, Anne Aaron, Ioannis Katsavounidis, Anush Moorthy, and Megha Manohara. *Toward A Practical Perceptual Video Quality Metric*. Tech. rep. Accessed: 2018-06-08. 2016.

[110] Zhicheng Li, Shiyin Qin, and Laurent Itti. "Visual attention guided bit allocation in video compression." In: *Image and Vision Computing* 29.1 (2011), pp. 1–14.

[111] Robert LiKamWa, Yunhui Hou, Julian Gao, Mia Polansky, and Lin Zhong. "RedEye: Analog ConvNet Image Sensor Architecture for Continuous Mobile Vision." In: *Proceedings of the 43nd Annual International Symposium on Computer Architecture (ISCA)*. Seoul, Korea: ACM, 2016.

[112] Robert LiKamWa, Zhen Wang, Aaron Carroll, Felix Xiaozhu Lin, and Lin Zhong. "Draining Our Glass: An Energy and Heat Characterization of Google Glass." In: *Proceedings of 5th Asia-Pacific Workshop on Systems*. APSys '14. Beijing, China: ACM, 2014.

[113] Luyang Liu, Ruiguang Zhong, Wuyang Zhang, Yunxin Liu, Jiansong Zhang, Lintao Zhang, and Marco Gruteser. "Cutting the Cord: Designing a High-quality Untethered VR System with Low Latency Remote Rendering." In: *Proceedings of the 16th Annual International Conference on Mobile Systems, Applications, and Services*. MobiSys '18. Munich, Germany: ACM, 2018, pp. 68–80. ISBN: 978-1-4503-5720-3. DOI: 10.1145/3210240.3210313. URL: http://doi.acm.org/10.1145/3210240.3210313.

[114] Peng Liu, Jongwon Yoon, Lance Johnson, and Suman Banerjee. "Greening the Video Transcoding Service with Low-Cost Hardware Transcoders." In: *2016 USENIX Annual Technical Conference (USENIX ATC 16)*. Denver, CO: USENIX Association, 2016, pp. 407–419. ISBN: 978-1-931971-30-0. URL: https://www.usenix.org/conference/atc16/technical-sessions/presentation/liu.

[115] Wen-Chih Lo, Ching-Ling Fan, Jean Lee, Chun-Ying Huang, Kuan-Ta Chen, and Cheng-Hsin Hsu. "360&Deg; Video Viewing Dataset in Head-Mounted Virtual Reality." In: *Proceedings of the 8th ACM on Multimedia Systems Conference*. MMSys'17. Taipei, Taiwan: ACM, 2017, pp. 211–216. ISBN: 978-1-4503-5002-0. DOI: 10.1145/3083187.3083219. URL: http://doi.acm.org/10.1145/3083187.3083219.

[116] Andrea Lottarini, Alex Ramirez, Joel Coburn, Martha A. Kim, Parthasarathy Ranganathan, Daniel Stodolsky, and Mark Wachsler. "Vbench: Benchmarking Video Transcoding in the Cloud." In: *Proceedings of the Twenty-Third International Conference on Architectural Support for Programming Languages and Operating Systems*. ASPLOS '18. Williamsburg, VA, USA: ACM, 2018, pp. 797–809. ISBN: 978-1-4503-4911-6. DOI: 10.1145/3173162.3173207. URL: http://doi.acm.org/10.1145/3173162.3173207.

[117] Yao Lu, Aakanksha Chowdhery, and Srikanth Kandula. "Optasia: A Relational Platform for Efficient Large-Scale Video Analytics." In: *Proceedings of the Seventh ACM Symposium on Cloud Computing*. SoCC '16. Santa Clara, CA, USA: ACM, 2016, pp. 57–70. DOI: 10.1145/2987550.2987564. URL: http://doi.acm.org/10.1145/2987550.2987564.

[118] V. Lyudvichenko, M. Erofeev, Y. Gitman, and D. Vatolin. "A semiautomatic saliency model and its application to video compression." In: *2017 13th IEEE International Conference on Intelligent Computer Communication and Processing (ICCP)*. Sept. 2017, pp. 403–410. DOI: 10.1109/ICCP.2017.8117038.

[119] Ziyang Ma, Kaiming He, Yichen Wei, Jian Sun, and Enhua Wu. "Constant time weighted median filtering for stereo matching and beyond." In: *ICCV* (2013).

[120] Ikuo Magaki, Moein Khazraee, Luis Vega Gutierrez, and Michael Bedford Taylor. "ASIC Clouds: Specializing the Datacenter." In: *Proceedings of the 43rd International Symposium on Computer Architecture*. ISCA '16. Seoul, Republic of Korea: IEEE Press, 2016, pp. 178–190. ISBN: 978-1-4673-8947-1. DOI: 10.1109/ISCA.2016.25. URL: https://doi.org/10.1109/ISCA.2016.25.

[121] Megha Manohara, Anush Moorthy, Jan De Cock, Ioannis Katsavounidis, and Anne Aaron. *Optimized shot-based encodes: Now Streaming!* Tech. rep. Accessed: 2018-06-08. 2018.

[122] Markus Mathias, Rodrigo Benenson, Marco Pedersoli, and Luc Van Gool. "Face Detection without Bells and Whistles." English. In: *ECCV 2014*. 2014. ISBN: 978-3-319-10592-5. DOI: 10.1007/978-3-319-10593-2_47.

[123] Amrita Mazumdar, Armin Alaghi, Jonathan T. Barron, David Gallup, Luis Ceze, Mark Oskin, and Steven M. Seitz. "A Hardware-friendly Bilateral Solver for Real-time Virtual Reality Video." In: *Proceedings of High Performance Graphics*. HPG '17. Los Angeles, California: ACM, July 2017, 13:1–13:10. ISBN: 978-1-4503-5101-0. DOI: 10.1145/3105762.3105772. URL: http://doi.acm.org/10.1145/3105762.3105772.

[124] Amrita Mazumdar, Brandon Haynes, Magda Balazinska, Luis Ceze, Alvin Cheung, and Mark Oskin. "Perceptual Compression for Video Storage and Processing Systems." In: *Proceedings of the ACM Symposium on Cloud Computing*. SoCC '19. Santa Cruz, CA, USA: ACM, 2019, pp. 179–192. ISBN: 978-1-4503-6973-2. DOI: 10.1145/3357223.3362725. URL: http://doi.acm.org/10.1145/3357223.3362725.

[125] Amrita Mazumdar, Thierry Moreau, Sung Kim, Meghan Cowan, Armin Alaghi, Luis Ceze, Mark Oskin, and Visvesh Sathe. "Exploring computation-communication tradeoffs in camera systems." In: *2017 IEEE International Symposium on Workload Characterization (IISWC)*. IEEE, Oct. 2017, pp. 177–186. DOI: 10.1109/IISWC.2017.8167775. URL: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=8167775&isnumber=8167743.

[126] Mediakix. *The Facebook Video Statistics Everyone Needs to Know*. http://mediakix.com/2016/08/facebook-video-statistics-everyone-needs-know/. 2016.

[127] Dongbo Min, Sunghwan Choi, Jiangbo Lu, Bumsub Ham, Kwanghoon Sohn, and Minh N. Do. "Fast global image smoothing based on weighted least squares." In: *Transactions on Image Processing* (2014).

[128] K. Misra, A. Segall, M. Horowitz, S. Xu, A. Fuldseth, and M. Zhou. "An Overview of Tiles in HEVC." In: *IEEE Journal of Selected Topics in Signal Processing* 7.6 (Dec. 2013), pp. 969–977. ISSN: 1932-4553. DOI: 10.1109/JSTSP.2013.2271451.

[129] Kiran Misra, Andrew Segall, Michael Horowitz, Shilin Xu, Arild Fuldseth, and Minhua Zhou. "An overview of tiles in HEVC." In: *IEEE journal of selected topics in signal processing* 7.6 (2013), pp. 969–977.

[130] T. Moreau, J. San Miguel, M. Wyse, J. Bornholt, A. Alaghi, L. Ceze, N. Enright Jerger, and A. Sampson. "A Taxonomy of General Purpose Approximate Computing Techniques." In: *IEEE Embedded Systems Letters* 10.1 (Mar. 2018), pp. 2–5. ISSN: 1943-0663. DOI: 10.1109/LES.2017.2758679.

[131] Thierry Moreau, Adrian Sampson, Luis Ceze, and Mark Oskin. "Approximating to the Last Bit." In: *Workshop on Approximate Computing Across the Stack (WAX)* (2016).

[132] Thierry Moreau, Mark Wyse, Jacob Nelson, Adrian Sampson, Hadi Esmaeilzadeh, Luis Ceze, and Mark Oskin. "SNNAP: Approximate Computing on Programmable SoCs via Neural Acceleration." In: *International Symposium on High-Performance Computer Architecture (HPCA)*. Feb. 2015.

[133] Debargha Mukherjee, Jingning Han, Jim Bankoski, Ronald Bultje, Adrian Grange, John Koleszar, Paul Wilkins, and Yaowu Xu. "A technical overview of VP9 - The latest open-source video codec." In: *SMPTE* 124.1 (2015), pp. 44–54.

[134] Subramanian Muralidhar et al. "f4: Facebook's Warm BLOB Storage System." In: *11th USENIX Symposium on Operating Systems Design and Implementation (OSDI 14)*. Broomfield, CO: USENIX Association, 2014, pp. 383–398. ISBN: 978-1-931971-16-4. URL: https://www.usenix.org/conference/osdi14/technical-sessions/presentation/muralidhar.

[135] Saman Naderiparizi, Yi Zhao, James Youngquist, Alanson P. Sample, and Joshua R. Smith. "Self-localizing Battery-free Cameras." In: *Proceedings of the 2015 ACM International Joint Conference on Pervasive and Ubiquitous Computing*. UbiComp '15. ACM, 2015. ISBN: 978-1-4503-3574-4.

[136] "New virtual reality video takes you up close with elephants." In: *National Geographic* (Dec. 2018). URL: https://www.nationalgeographic.com/animals/2018/12/okavango-wilderness-360-elephant-herd/.

[137] A. Nilchi, J. Aziz, and R. Genov. "Focal-Plane Algorithmically-Multiplying CMOS Computational Image Sensor." In: *IEEE Journal of Solid-State Circuits* 44.6 (June 2009), pp. 1829–1839. ISSN: 0018-9200. DOI: 10.1109/JSSC.2009.2016693.

[138] S. Nissen. *Implementation of a Fast Artificial Neural Network Library (fann)*. Tech. rep. http://fann.sf.net. Department of Computer Science University of Copenhagen (DIKU), 2003.

[139] Andrey Norkin, Jan De Cock, Aditya Mavlankar, and Anne Aaron. *More Efficient Mobile Encodes for Netflix Downloads*. Tech. rep. Accessed: 2018-06-08. 2018.

[140] *Nvidia Video codec*. https://developer.nvidia.com/nvidia-video-codec-sdk.

[141] NVIDIA. *Jetson TX2*. 2019. URL: https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/jetson-tx2/.

[142] NVIDIA. *Tegra Linux Driver Package*. 2019. URL: https://docs.nvidia.com/jetson/l4t/index.html.

[143] Nvidia. *Nvidia Tegra*. http://www.nvidia.com/object/tegra.html. Accessed: 2017-06-06.

[144] Oculus. *Oculus Rift*. Accessed: 2017-06-15. 2017.

[145] Virginia E Ogle and Michael Stonebraker. "Chabot: Retrieval from a relational database of images." In: *Computer* 28.9 (1995), pp. 40–48.

[146] Chris Olah, Arvind Satyanarayan, Ian Johnson, Shan Carter, Ludwig Schubert, Katherine Ye, and Alexander Mordvintsev. "The Building Blocks of Interpretability." In: *Distill* (2018). https://distill.pub/2018/building-blocks. DOI: 10.23915/distill.00010.

[147] OpenCores. *Openmsp430*. http://opencores.org/project,openmsp430. Accessed: 2017-06-06.

[148] C.P. Papageorgiou, M. Oren, and T. Poggio. "A general framework for object detection." In: *Computer Vision, 1998. Sixth International Conference on*. Jan. 1998, pp. 555–562. DOI: 10.1109/ICCV.1998.710772.

[149] Netflix Party. *NetflixParty: A new way to watch Netflix together*. Accessed: 2020-04-17. 2020.

[150] Anjul Patney. "Perceptual insights into foveated virtual reality." In: *NVIDIA GPU Technology Conference*. 2017.

[151] S. Peleg, M. Ben-Ezra, and Y. Pritch. "Omnistereo: panoramic stereo imaging." In: *PAMI* (2001). ISSN: 0162-8828.

[152] Boris Teodorovich Polyak. "Some methods of speeding up the convergence of iteration methods." In: *USSR Computational Mathematics and Mathematical Physics* 4.5 (1964).

[153] Alex Poms, Will Crichton, Pat Hanrahan, and Kayvon Fatahalian. "Scanner: Efficient Video Analysis at Scale." In: *ACM Trans. Graph.* 36.4 (2018).

[154] Wajahat Qadeer, Rehan Hameed, Ofer Shacham, Preethi Venkatesan, Christos Kozyrakis, and Mark A. Horowitz. "Convolution Engine: Balancing Efficiency & Flexibility in Specialized Computing." In: *SIGARCH Comput. Archit. News* 41.3 (June 2013), pp. 24–35. ISSN: 0163-5964. DOI: 10.1145/2508148.2485925. URL: http://doi.acm.org/10.1145/2508148.2485925.

[155] Wajahat Qadeer, Rehan Hameed, Ofer Shacham, Preethi Venkatesan, Christos Kozyrakis, and Mark A. Horowitz. "Convolution Engine: Balancing Efficiency and Flexibility in Specialized Computing." In: *Proceedings of the 40th Annual International Symposium on Computer Architecture*. Tel-Aviv, Israel: ACM, 2013, pp. 24–35. ISBN: 978-1-4503-2079-5. DOI: 10.1145/2485922.2485925. URL: http://doi.acm.org/10.1145/2485922.2485925.

[156] Qualcomm. *Snapdragon*. https://www.qualcomm.com/products/snapdragon/processors. Accessed: 2017-06-06.

[157] Jonathan Ragan-Kelley, Connelly Barnes, Andrew Adams, Sylvain Paris, Frédo Durand, and Saman Amarasinghe. "Halide: a language and compiler for optimizing parallelism, locality, and recomputation in image processing pipelines." In: *ACM SIGPLAN Notices* 48.6 (2013), pp. 519–530.

[158] Brandon Reagen, Paul Whatmough, Robert Adolf, Saketh Rama, Hyunkwang Lee, Sae Kyu Lee, José Miguel Hernández-Lobato, Gu-Yeon Wei, and David Brooks. "Minerva: Enabling Low-Power, Highly-Accurate Deep Neural Network Accelerators." In: *International Symposium on Computer Architecture (ISCA)*. June 18, 2016. URL: http://vlsiarch.eecs.harvard.edu/wp-content/uploads/2016/05/reagen_isca16.pdf.

[159] Joseph Redmon and Ali Farhadi. "YOLO9000: Better, Faster, Stronger." In: *CVPR*. 2017, pp. 6517–6525.

[160] R. Rithe, P. Raina, N. Ickes, S. V. Tenneti, and A. P. Chandrakasan. "Reconfigurable processor for energy-scalable computational photography." In: *2013 IEEE International Solid-State Circuits Conference Digest of Technical Papers*. Feb. 2013, pp. 164–165. DOI: 10.1109/ISSCC.2013.6487683.

[161] Jihoon Ryoo, Kiwon Yun, Dimitris Samaras, Samir R. Das, and Gregory Zelinsky. "Design and Evaluation of a Foveated Video Streaming Service for Commodity Client Devices." In: *Proceedings of the 7th International Conference on Multimedia Systems*. MMSys '16. Klagenfurt, Austria: ACM, 2016, 6:1–6:11. ISBN: 978-1-4503-4297-1. DOI: 10.1145/2910017.2910592. URL: http://doi.acm.org/10.1145/2910017.2910592.

[162] Ferdinando S Samaria and Andy C Harter. "Parameterisation of a stochastic model for human face identification." In: *Applications of Computer Vision, 1994., Proceedings of the Second IEEE Workshop on*. IEEE. 1994.

[163] A.P. Sample, D.J. Yeager, P.S. Powledge, A.V. Mamishev, and J.R. Smith. "Design of an RFID-Based Battery-Free Programmable Sensing Platform." In: *Instrumentation and Measurement, IEEE Transactions on* 57.11 (Nov. 2008), pp. 2608–2615. ISSN: 0018-9456. DOI: 10.1109/TIM.2008.925019.

[164] Adrian Sampson, Werner Dietl, Emily Fortuna, Danushen Gnanapragasam, Luis Ceze, and Dan Grossman. "EnerJ: Approximate data types for safe and general low-power computation." In: *ACM SIGPLAN Notices*. Vol. 46. 6. ACM. 2011, pp. 164–174.

[165] Samsung. *Gear VR*. http://www.samsung.com/us/explore/gear-vr/. Accessed: 2017-06-06.

[166] M. Satyanarayanan. "Pervasive computing: vision and challenges." In: *IEEE Personal Communications* 8.4 (Aug. 2001), pp. 10–17. ISSN: 1070-9916. DOI: 10.1109/98.943998.

[167] Daniel Scharstein and Richard Szeliski. "A taxonomy and evaluation of dense two-frame stereo correspondence algorithms." In: *International journal of computer vision* (2002).

[168] Haichen Shen, Seungyeop Han, Matthai Philipose, and Arvind Krishnamurthy. "Fast Video Classification via Adaptive Cascading of Deep Models." In: *2017 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017, Honolulu, HI, USA, July 21-26, 2017*. 2017, pp. 2197–2205.

[169] Heung-Yeung Shum, King-To Ng, and Shing-Chow Chan. "A virtual reality system using the concentric mosaic: construction, rendering, and data compression." In: *IEEE Transactions on Multimedia* (2005).

[170] Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. "Deep Inside Convolutional Networks: Visualising Image Classification Models and Saliency Maps." In: *ICLR WOrkshop* (2013).

[171] V. Sitzmann, A. Serrano, A. Pavel, M. Agrawala, D. Gutierrez, B. Masia, and G. Wetzstein. "Saliency in VR: How Do People Explore Virtual Environments?" In: *IEEE Transactions on Visualization and Computer Graphics* 24.4 (Apr. 2018), pp. 1633–1642. ISSN: 1077-2626. DOI: 10.1109/TVCG.2018.2793599.

[172] Phillip Stanley-Marbell, Virginia Estellers, and Martin Rinard. "Crayon: Saving Power Through Shape and Color Approximation on Next-generation Displays." In: *Proceedings of the Eleventh European Conference on Computer Systems*. EuroSys '16. London, United Kingdom: ACM, 2016, 11:1–11:17. ISBN: 978-1-4503-4240-7. DOI: 10.1145/2901318.2901347. URL: http://doi.acm.org/10.1145/2901318.2901347.

[173] Phillip Stanley-Marbell and Martin Rinard. "Perceived-Color Approximation Transforms for Programs that Draw." In: *IEEE Micro* 38.4 (2018), pp. 20–29.

[174] Gary J Sullivan, Jens-Rainer Ohm, Woo-Jin Han, and Thomas Wiegand. "Overview of the high efficiency video coding (HEVC) standard." In: *IEEE Transactions on circuits and systems for video technology* 22.12 (2012), pp. 1649–1668.

[175] Deqing Sun, Stefan Roth, and Michael J. Black. "Secrets of optical flow estimation and their principles." In: *CVPR* (2010). ISSN: 1063-6919.

[176] Linpeng Tang, Qi Huang, Amit Puntambekar, Ymir Vigfusson, Wyatt Lloyd, and Kai Li. "Popularity Prediction of Facebook Videos for Higher Quality Streaming." In: *2017 USENIX Annual Technical Conference (USENIX ATC 17)*. Santa Clara, CA: USENIX Association, 2017, pp. 111–123. ISBN: 978-1-931971-38-6. URL: https://www.usenix.org/conference/atc17/technical-sessions/presentation/tang.

[177] Teradek. *Sphere - Real-time 360 Monitoring and Live Streaming*. http://teradek.com/collections/sphere-family/. Accessed: 2017-06-06.

[178] Theano Development Team. "Theano: A Python framework for fast computation of mathematical expressions." In: *arXiv e-prints* abs/1605.02688 (May 2016). URL: http://arxiv.org/abs/1605.02688.

[179] M. Tikekar, V. Sze, and A. P. Chandrakasan. "A Fully Integrated Energy-Efficient H.265/HEVC Decoder With eDRAM for Wearable Devices." In: *IEEE Journal of Solid-State Circuits* 53.8 (Aug. 2018), pp. 2368–2377. DOI: 10.1109/JSSC.2018.2837124.

[180] Mehul Tikekar. "Energy-efficient video decoding using data statistics." PhD thesis. Massachusetts Institute of Technology: Massachusetts Institute of Technology, 2017.

[181] Artem Vasilyev, Nikhil Bhagdikar, Ardavan Pedram, Stephen Richardson, Shahr Kvatinsky, and Mark Horowitz. "Evaluating Programmable Architectures for Imaging and Vision Applications." In: *2016 49th IEEE/ACM International Symposium on Microarchitecture, 2016. MICRO-49*. Oct. 2016.

[182] Michael Vernick, Chitra Venkatramani, and Tzi-cker Chiueh. "Adventures in Building the Stony Brook Video Server." In: *Proceedings of the Fourth ACM International Conference on Multimedia*. MULTIMEDIA '96. Boston, Massachusetts, USA: ACM, 1996, pp. 287–295. ISBN: 0-89791-871-1. DOI: 10.1145/244130.244230. URL: http://doi.acm.org.offcampus.lib.washington.edu/10.1145/244130.244230.

[183] Videostitch. *Vahana VR*. http://www.video-stitch.com/live-vr/. Accessed: 2017-06-06.

[184] Paul Viola and Michael J. Jones. "Robust real-time face detection." In: *International journal of computer vision* 57.2 (2004). (Visited on 09/23/2015).

[185] Zhou Wang and Alan C. Bovik. "Foveated Image and Video Coding." In: *Digital Video Image Quality and Perceptual Coding*. Ed. by H.R. Wu and K.R. Rao. CRC Press, 2006. Chap. 14.

[186] Zhou Wang, Eero P Simoncelli, and Alan C Bovik. "Multiscale structural similarity for image quality assessment." In: *Signals, Systems and Computers, 2004. Conference Record of the Thirty-Seventh Asilomar Conference on*. Vol. 2. Ieee. 2003.

[187] Eric Whitmire, Laura Trutoiu, Robert Cavin, David Perek, Brian Scally, James Phillips, and Shwetak Patel. "EyeContact: Scleral Coil Eye Tracking for Virtual Reality." In: *Proceedings of the 2016 ACM International Symposium on Wearable Computers*. ISWC '16. Heidelberg, Germany: ACM, 2016, pp. 184–191. ISBN: 978-1-4503-4460-9. DOI: 10.1145/2971763.2971771. URL: http://doi.acm.org/10.1145/2971763.2971771.

[188] Thomas Wiegand, Gary J. Sullivan, Gisle Bjntegaard, and Ajay Luthra. "Overview of the H.264/AVC video coding standard." In: *TCSVT* 13.7 (2003), pp. 560–576.

[189] Clifford Wolf. *Yosys Open SYnthesis Suite*. http://www.clifford.at/yosys/.

[190] Lisa Wu, Andrea Lottarini, Timothy K. Paine, Martha A. Kim, and Kenneth A. Ross. "Q100: The Architecture and Design of a Database Processing Unit." In: *Proceedings of the 19th International Conference on Architectural Support for Programming Languages and Operating Systems*. ASPLOS '14. Salt Lake City, Utah, USA: ACM, 2014, pp. 255–268. ISBN: 978-1-4503-2305-5. DOI: 10.1145/2541940.2541961. URL: http://doi.acm.org/10.1145/2541940.2541961.

[191] Chenhao Xie, Xin Fu, and Shuaiwen Song. "Perception-oriented 3D rendering approximation for modern graphics processors." In: *2018 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE. 2018, pp. 362–374.

[192] Ran Xu, Jinkyu Koo, Rakesh Kumar, Peter Bai, Subrata Mitra, Sasa Misailovic, and Saurabh Bagchi. "Videochef: efficient approximation for streaming video processing pipelines." In: *USENIX ATC*. 2018, pp. 43–56.

[193] Zhisheng Yan and Chang Wen Chen. "Too Many Pixels to Perceive: Subpixel Shutoff for Display Energy Reduction on OLED Smartphones." In: *Proceedings of the 25th ACM International Conference on Multimedia*. MM '17. Mountain View, California, USA: ACM, 2017, pp. 717–725. ISBN: 978-1-4503-4906-2. DOI: 10.1145/3123266.3123344. URL: http://doi.acm.org/10.1145/3123266.3123344.

[194] Q. Yang. "Hardware-Efficient Bilateral Filtering for Stereo Matching." In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 36.5 (May 2014). ISSN: 0162-8828. DOI: 10.1109/TPAMI.2013.186.

[195] Qingxiong Yang. "Stereo matching using tree filtering." In: *PAMI* (2015).

[196] Matthew D. Zeiler and Rob Fergus. "Visualizing and Understanding Convolutional Networks." In: *ECCV*. 2014.

[197] Haibo Zhang, Prasanna Venkatesh Rengasamy, Shulin Zhao, Nachiappan Chidambaram Nachiappan, Anand Sivasubramaniam, Mahmut T Kandemir, Ravi Iyer, and Chita R Das. "Race-to-sleep+ content caching+ display caching: a recipe for energy-efficient video streaming on handhelds." In: *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture*. ACM. 2017, pp. 517–531.

[198] Haibo Zhang, Prasanna Venkatesh Rengasamy, Shulin Zhao, Nachiappan Chidambaram Nachiappan, Anand Sivasubramaniam, Mahmut T. Kandemir, Ravi Iyer, and Chita R. Das. "Race-to-sleep + Content Caching + Display Caching: A Recipe for Energy-efficient Video Streaming on Handhelds." In: *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture*. MICRO-50 '17. Cambridge, Massachusetts: ACM, 2017, pp. 517–531. ISBN: 978-1-4503-4952-9. DOI: 10.1145/3123939.3123948. URL: http://doi.acm.org/10.1145/3123939.3123948.

[199] Haoyu Zhang, Ganesh Ananthanarayanan, Peter Bodik, Matthai Philipose, Paramvir Bahl, and Michael J. Freedman. "Live Video Analytics at Scale with Approximation and Delay-Tolerance." In: *NSDI*. 2017. URL: https://www.usenix.org/conference/nsdi17/technical-sessions/presentation/zhang.

[200] Haoyu Zhang, Ganesh Ananthanarayanan, Peter Bodik, Matthai Philipose, Paramvir Bahl, and Michael J. Freedman. "Live Video Analytics at Scale with Approximation and Delay-Tolerance." In: *14th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2017, Boston, MA, USA, March 27-29, 2017*. 2017, pp. 377–392.

[201] Qi Zhang, Li Xu, and Jiaya Jia. "100+ times faster weighted median filter (WMF)." In: *CVPR* (2014).

[202] D. Zhou et al. "An 8K H.265/HEVC Video Decoder Chip With a New System Pipeline Design." In: *IEEE Journal of Solid-State Circuits* 52.1 (Jan. 2017), pp. 113–126. DOI: 10.1109/JSSC.2016.2616362.

[203] Jihan Zhu and Peter Sutton. "FPGA Implementations of Neural Networks – A Survey of a Decade of Progress." English. In: *Field Programmable Logic and Application*. Ed. by Peter Y. K. Cheung and GeorgeA. Constantinides. Vol. 2778. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2003. ISBN: 978-3-540-40822-2.

[204] Luisa M. Zintgraf, Taco Cohen, Tameem Adel, and Max Welling. "Visualizing Deep Neural Network Decisions: Prediction Difference Analysis." In: *ICLR* (2017).

[205] Fabio Zund, Yael Pritch, Alexander Sorkine-Hornung, Stefan Mangold, and Thomas Gross. "Content-aware compression using saliency-driven image retargeting." In: *Image Processing (ICIP), 2013 20th IEEE International Conference on*. IEEE. 2013, pp. 1845–1849.

COLOPHON

This dissertation was typeset using the typographical look-and-feel `classicthesis` developed by André Miede and Ivo Pletikosić. The full source code for this dissertation is available online:
https://github.com/amritamaz/thesis

Final Version as of June 18, 2020 (classicthesis v4.6).